SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
V A R A Ž D I N

Anđelko Spevec

Oracle PL/SQL

DIPLOMSKI RAD

Varaždin, 2014.

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
V A R A Ž D I N

Anđelko Spevec
Matični broj: 41753/12-R
Studij: Baze podataka i baze znanja

Oracle PL/SQL

DIPLOMSKI RAD

Mentor:
Doc.dr.sc. Markus Schatten

Varaždin, lipanj 2014.

Sadržaj

1. Uvod

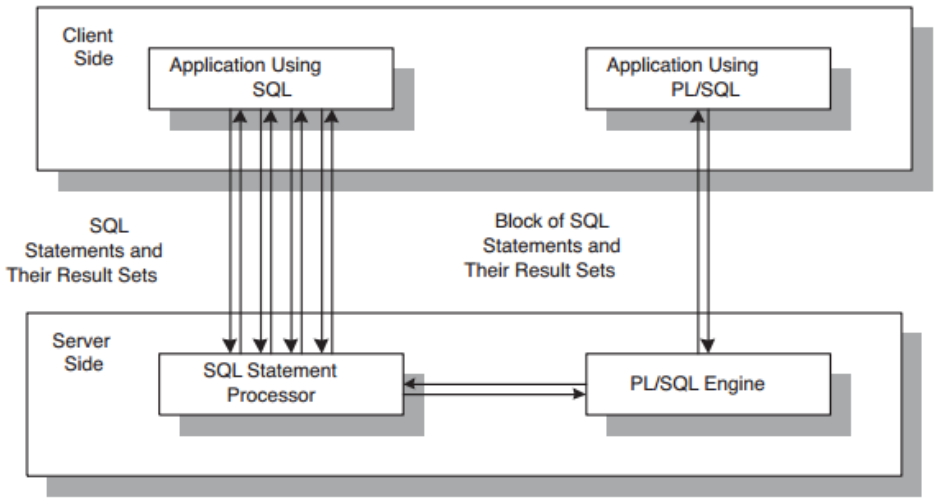
U današnje vrijeme kad gotovo svaka aplikacija koristi bazu podataka za pohranu svojih podataka od velike je važnosti iskoristiti te podatke što bolje. Povezanost između aplikacija i baze podataka se može obaviti koristeći standardne SQL DML (*eng. Data Manipulation Language*) naredbe, ali svaki od dohvata podataka iz baze zahtjeva prijenos putem mreže, pretvorbu podataka iz tipa podataka iz baze u tip podataka u programskom jeziku što oduzima i vrijeme i resurse. Zato PL/SQL omogućava da se više naredbi odradi unutar jednog bloka, odnosno da se podatci i obrade koristeći operatore, petlje i provjere kao u ostalim programskim jezicima. Omogućava se i prihvat i obrada iznimaka unutar samog bloka što olakšava sam rad s podatcima. PL/SQL aplikacije su prenosive i neovisne o operacijskom sustavu, te tako omogućavaju konzistentan rad na svim računalima.

U ovo radu će se obraditi mogućnosti PL/SQL jezika koji se koristi za Oracle baze podataka te ih prikazati na primjerima. Nakon toga će se mogućnosti PL/SQL-a usporediti s mogućnostima proceduralnog jezika za PostgreSQL bazu podataka (PL/pg SQL) te proceduralnog jezika za Microsoft i Sybase bazu podataka (T-SQL).

2. Uvod u PL/SQL

PL/SQL (*eng. Procedural Language/Structured Query Language*) je proširenje proceduralnog jezika za SQL u Oracle bazama podataka. To je kombinacija SQL-a s proceduralnim osobinama programskih jezika kao što su petlja, selekcija, iteracija, deklariranje varijabli, strukture, upravljanje greškama itd. Može koristiti sve SQL naredbe, funkcije, operatore i tipove podataka koji su dostupni u SQL-u. [1]

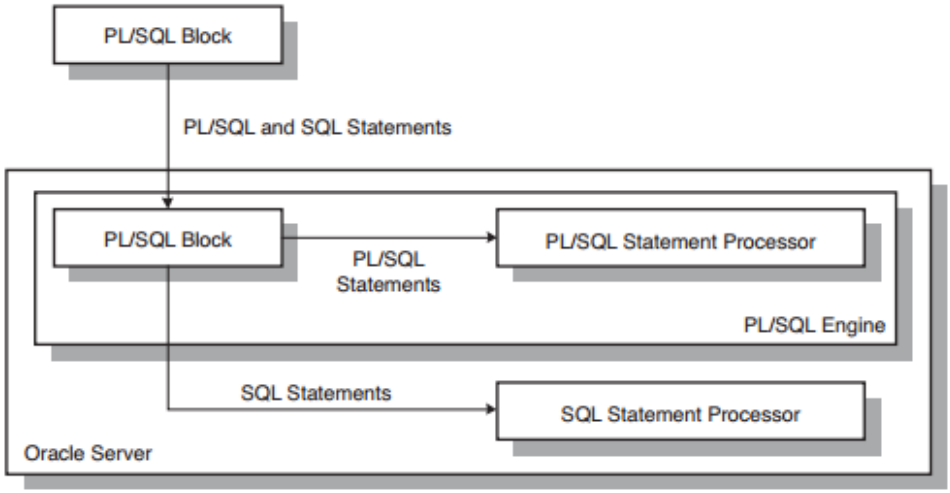
Još jedna od prednosti PL/SQL-je da se može grupirati više SQL naredbi te ih poslati prema bazi kao jedan upit te se tako ubrzava aplikacija i smanjuje mrežni promet. Inače svaka SQL naredba koja se izvršava putuje od klijenta do baze podataka na poslužitelju, te nakon obrade vraća rezultat korisniku. Za svaku naredbu je isti postupak, pa se kod više naredbi povećava i mrežni promet jer svaki od njih uzrokuje dva prijenosa putem mreže. Ako je više takvih naredbi unutar PL/SQL bloka oni se šalju kao jedna cjelina te se rezultat čitavog bloka također vrati kao jedna cjelina. [2] Na slici 1 je prikazana komunikacija između klijenta i poslužitelja.



Slika 1. Komunikacija između klijenta i poslužitelja za SQL i PL/SQL[3]

Razlika između SQL-a i PL/SQL-a je i u načinu izvođenja. Za izvođenje PL/SQL bloka je zadužen PL/SQL stroj koji izvršava PL/SQL izraze dok SQL naredbe šalje na procesor zadužen za obradu SQL izraza (*eng. SQL statement processor*). Ako se PL/SQL stroj nalazi na poslužitelju, obrada se ostvaruje kao što je prikazano na slici 2.

[1] Oracle (2009) - Oracle Database PL/SQL Language Reference 11g Release 1, *Tight Integration with SQL*, URL: http://docs.oracle.com/cd/B28359_01/appdev.111/b28370/overview.htm#CJAGBBAD, dostupno 14.06.2014.
[2] Rosenzweig B., Rakhimov E. (2009) – *Oracle PL/SQL by Example Fourth Edition*, str. 2.
[3] Rosenzweig B., Rakhimov E. (2009) – *Oracle PL/SQL by Example Fourth Edition*, str. 4.



Slika 2. Obrada PL/SQL bloka[4]

Ako se PL/SQL stroj nalazi na klijentskoj strani tada obrada PL/SQL-a se ostvaruje na klijentskoj strani dok se SQL naredbe koje se nalaze u bloku šalju na server na daljnju obradu. Uz to aplikacije pisane u PL/SQL-u su neovisne o operacijskom sustavu i platformi na kojoj se nalazi baza.

PL/SQL kombinira mogućnosti SQL-a za manipulaciju podataka te ih kombinira s mogućnostima obrade proceduralnih jezika. Tako omogućava deklaraciju varijabli i konstanti, kao i kontrolu toka programa, potprograme i obradu iznimki. Pisanje potprograma omogućava smanjivanje kompleksnosti programa podjelom u manje, više razumljive potprograme koji se kasnije mogu koristiti. [5]

2.1. PL/SQL blok[6]

Najosnovnija struktura PL/SQL-a je blok, te je uobičajeno da jedan blok predstavlja jednu logičku cjelinu koja se izvršava. PL/SQL blok se sastoji od tri dijela, a to su deklarativni dio, izvršni dio i dio za obradu iznimaka. Svaki dio započinje svojom ključnom riječi, a blok završava ključnom riječi END.

DECLARE
    Izrazi za deklariranje varijabli
BEGIN
    Izrazi za izvršavanje

[4] Rosenzweig B., Rakhimov E. (2009) – *Oracle PL/SQL by Example Fourth Edition*, str. 3.
[5] Oracle (2009) - Oracle Database PL/SQL Language Reference 11g Release 1, *PL/SQL Blocks*, URL: http://docs.oracle.com/cd/B28359_01/appdev.111/b28370/overview.htm#i8859, dostupno 14.06.2014.
[6] TutorialsPoint (2014) - PL/SQL Tutorial, *PL/SQL - Basic Syntax*, URL: http://www.tutorialspoint.com /plsql/plsql_basic_syntax.htm, dostupno 14.06.2014.

```
EXCEPTION
    Izrazi za obradu iznimaka
END;
```

2.1.1. Deklarativni dio

Prvi dio PL/SQL bloka je deklarativni dio, te se u njemu definiraju varijable, konstante, kursori i slično. Sam dio započinje ključnom riječi DECLARE. Svaka varijabla ima naziv koji se dodjeljuje memorijskom prostoru kao i tip podataka koji određuje veličinu prostora i operacije koje se mogu obavljaju nad tom varijablom.

```
DECLARE
    ime_zaposlenika VARCHAR2(30);
    broj_sati INTEGER:
    cijena_po_satu CONSTANT DECIMAL(5,2) := 100.00;
```

2.1.2. Izvršni dio

Sljedeći dio PL/SQL bloka je izvršni dio koji započinje ključnom riječi BEGIN, te on sadrži izvršne naredbe.

```
DECLARE
    a INTEGER := 5;
    b INTEGER := 4;
    opseg INTEGER;
BEGIN
    opseg := 2 * a + 2 * b;
    DBMS_OUTPUT.PUT_LINE('Opseg je: ' || opseg);
END;

Opseg je: 18
```

No, u izvršnom bloku se inače izvršavaju složenije operacije, te ćemo njih opisivati i prolaziti u sljedećim poglavljima. Kod ovog djela bloka je važno napomenuti da je to jedini obavezan dio bloka, tako da mogu postojati blokovi samo sa izvršnim djelom. Izvršni dio se mora sastojati od barem jedne naredbe, ali moguće je da ta naredba bude prazna, odnosno *null*.

```
BEGIN
    null;
END;
```

U izvršnom djelu se još mogu koristiti i SQL naredbe te vrijednosti rezultata zapisivati u varijable deklarirane u deklarativnom djelu.

2.1.3. Dio za obradu iznimaka

Zadnji dio PL/SQL bloka je dio za obradu iznimaka te počinje ključnom riječi EXCEPTION. Sadrži naredbe koje se izvršavaju u slučaju da dođe do iznimke kod izvođenja u izvršnom djelu bloka.

```
DECLARE
        ime VARCHAR2(30);
BEGIN
        SELECT ime_zap
        INTO ime
        FROM zaposlenik
        WHERE id_zap = 11;
        DBMS_OUTPUT.PUT_LINE('Ime zaposlenika: ' || ime);
EXCEPTION
        WHEN NO_DATA_FOUND THEN
                DBMS_OUTPUT.PUT_LINE('Nema zaposlenika s tim identifikacijskim
          brojem');
END;

Nema zaposlenika s tim identifikacijskim brojem
```

Ovaj PL/SQL blok izvršava SQL naredbu u izvršnom djelu te zapisuje rezultat te naredbe u varijablu deklariranu u deklarativnom djelu bloka. U slučaju da SELECT INTO naredba ne vrati niti jedan red, pokreće se obrada iznimke NO_DATA_FOUND i ispisuje odgovarajuću poruku.

PL/SQL blokovi se mogu ugnježđivati unutar drugih blokova, te se tako unutar bloka može deklarirati drugi blok bilo gdje unutar izvršnog dijela. Blokove dijelimo još u dvije grupe, a to su imenovani i anonimni. Imenovani blokovi kao što su procedure, funkcije, okidači i paketi mogu biti spremljeni za kasnije korištenje. Oni se prevode u strojni kôd kod kreiranja i kod svake promjene, pa se kod pozivanja ne trebaju. Dok se anonimni blokovi prevode u strojni kôd prije izvršavanja, te kasnije ne mogu biti pozvani.

2.2. Prednosti PL/SQL-a

Uz sve prednosti PL/SQL-a koje su navedene postoje još[7]:

- Uska povezanost s SQL-om
- Provjera i obrada pogrešaka
- Veliki broj tipova podataka
- Razne strukture podataka

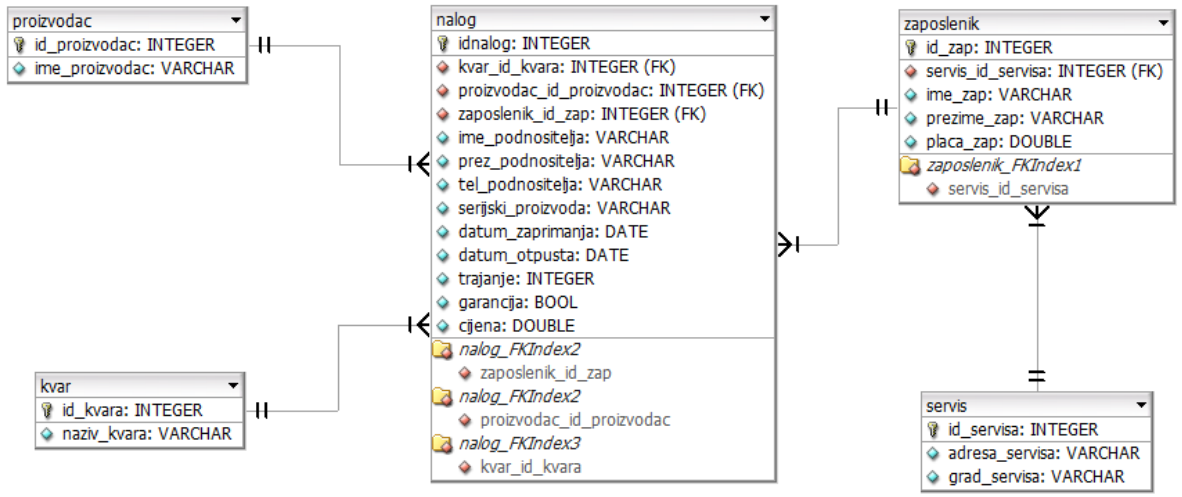
[7] TutorialsPoint (2014) - PL/SQL Tutorial, *PL/SQL - Overview*, URL: http://www.tutorialspoint.com/plsql/plsql_overview.htm, dostupno 14.06.2014.

- Omogućava funkcije i procedure
- Podržava objektno orijentirano programiranje
- Veliki broj predefiniranih SQL paketa
- Visoka razina sigurnosti
- Prijenosne aplikacije

3. ERA model baze podataka

Za potrebe ovog rada i zbog lakšeg objašnjavanja mogućnosti PL/SQL-a, kreirana je Oracle baza prema ERA modelu na slici 3. Baza je zamišljena kao mali servis laptopa, te se zapisuju nalozi odnosno uređaji zaprimljeni na servis. Tablice u bazi su:

- proizvodac – podatci o proizvođaču uređaja koji je zaprimljen na servis
- kvar – općenita podjela kvarova uređaja (npr. pregrijavanje, grafička kartica i slično)
- servis – podatci o lokaciji servisa (grad, adresa)
- zaposlenik – podatci o zaposlenicima servisa te njihova pripadnost određenoj poslovnici
- nalog – podatci o zaprimljenom uređaju, kvaru, proizvođaču, kontaktni podatci, vrijeme zaprimanja, osoba koja je zaprimila uređaj, da li je uređaj pod garancijom, cijena servisiranja i slično



Slika 3. ERA model baze podataka

Sama baza je kreirana na Oracle 11.1g sustavu za upravljanje bazama podataka te je za rad s njom, odnosno pisanje PL/SQL koda korišten Oracle SQL Developer.

4. Varijable i konstante u PL/SQL-u[8]

Varijable i konstante se u PL/SQL-u deklariraju u deklarativnom dijelu bloka, a koriste se u izvršnom dijelu. Tako se u deklarativnom dijelu mogu definirati varijable, konstante, kursori i slično. Svaka varijabla ima naziv koji se dodjeljuje memorijskom prostoru kao i tip podataka koji određuje veličinu prostora i operacije koje se mogu obavljati nad tom varijablom. Naziv varijable započinje slovom, te može sadržavati slova, brojeve, znakove ($,_,#) i mora biti manje od 30 znakova. Tip podatka također mora biti valjani PL/SQL tip podatka. Osnovna sintaksa za deklariranje varijable je sljedeća:

naziv_varijable [CONSTANT] tip_podataka [NOT NULL] [:= | DEFAULT inicijalna];

Iz gore navedene sintakse se vidi da je za varijablu obavezno napisati njezin naziv i tip podataka. Nakon naziva varijable se može navesti je li varijabla konstanta, tako da se prije tipa podataka napiše ključna riječ CONSTANT. Ako se navede da je varijabla konstanta, mora obavezno sadržavati i inicijalnu vrijednost koja se kasnije u bloku ne može mijenjati. Za varijablu se može definirati i da ne bude prazna odnosno da ne bude *null* koristeći ključnu riječ NOT NULL nakon tipa podatka. Kada se upiše ta ključna riječ, također mora postojati i inicijalna vrijednost koja se može mjenjati dalje unutar bloka. Za svaku varijablu inicijalno vrijedi NULL, odnosno dopušteno je da bude i prazna, odnosno *null*. Tako da se taj dio ne deklarira kod postavljanja same varijable. Inicijalna vrijednost varijable se postavlja koristeći operator pridruživanja := ili ključnu riječ DEFAULT. Deklaracija svake varijable završava s točka-zarez.

```
DECLARE
    ime_zaposlenika VARCHAR2(30);
    broj_sati INTEGER:
    cijena_po_satu CONSTANT DECIMAL(5,2) := 100.00;
```

Kao što je već spomenuto blokovi se mogu nalaziti jedni unutar drugih, te se javlja pitanje vidljivosti varijabli. Tako postoje dvije vidljivosti varijable, a to su lokalna i globalna vidljivosta. Globalno definirane varijable su definirane u vanjskom bloku, te su vidljive svim blokovima koji se nalaze unutar tog bloka. Dok su lokalno definirane varijable definirane u unutarnjem bloku i nisu dostupne za korištenje u vanjskom bloku.

```
DECLARE
    prvi_broj INTEGER := 11;
```

[8] TutorialsPoint (2014) - PL/SQL Tutorial, *PL/SQL - Variables*, URL: http://www.tutorialspoint.com/plsql/plsql_variable_types.htm, dostupno 14.06.2014.

```
        drugi_broj INTEGER := 12;
BEGIN
        DBMS_OUTPUT.PUT_LINE('Vanjski blok – prvi: ' || prvi_broj);
        DBMS_OUTPUT.PUT_LINE('Vanjski blok – drugi: ' || drugi_broj);
        --unutarnji blok
        DECLARE
                prvi_broj INTEGER := 13;
        BEGIN
                DBMS_OUTPUT.PUT_LINE('Unutarnji blok – prvi: ' || prvi_broj);
                DBMS_OUTPUT.PUT_LINE('Unutarnji blok – drugi: ' || drugi_broj);
        END;
END;

Vanjski blok – prvi: 11
Vanjski blok – drugi: 12
Unutarnji blok – prvi: 13
Unutarnji blok – drugi: 12
```

U ovom bloku su definirane dvije varijable u vanjskom bloku, odnosno te su varijable globalno vidljive. Dok je u drugom, unutarnjem bloku definirana treća varijabla koja je vidljiva samo unutar tog bloka.

Kada se unutar vanjskog bloka ispisuju dvije varijable one ispisuju vrijednosti 11 i 12. U unutarnjem bloku se deklarira nova varijabla pod istim imenom kao i u vanjskom bloku samo joj je inicijalna vrijednost 13. Pa kod ponovnog ispisa prve varijable ispisuje ona koja je lokalno definirana, odnosno ispisuje se vrijednost 13. Dok se kod ispisa druge varijable ispisuje globalno definirana varijabla, odnosno vrijednost 12.

U gornjem bloku je korištena DBMS_OUTPUT.PUT_LINE naredba koja je dio DBMS_OUTPUT paketa koji načešće služi za prikaz poruka i izvještaja. Dok naredbe PUT i PUT_LINE mogu stavljati informacije u međuspremnik, kako bi se te iste mogle pročitati od strane drugog okidača, procedure ili paketa.

Uz to korišteni sui komentari unutar bloka, koji se u PL/SQL blokovima obilježavaju na dva načina. Prvi način je koristeći dvije crtice -- koje značavaju da je taj redak komentar, a drugi način je da komentar započinje znakom /* te se nastavlja sve do prvog pojavljivanja znaka */ što znači da se može pisati preko više redaka.

5. Tipovi podataka u PL/SQL-u[9]

Svaka varijabla i parametar moraju imati valjane tipove podataka koji definiraju način spremanja, vrijednosti koje mogu poprimiti, operacije koje se mogu izvoditi nad njima i ograničenja koja se primjenjuju. PL/SQL ima velik broj predefiniranih tipova podataka i njihovih podtipova te dopušta stvaranje vlastitih PL/SQL podtipova. Podtip je podskup drugog tipa podataka koji se zove osnovni tip, a podtip ima sve iste operacija kao i osnovni ali nema isti raspon vrijednosti koje može poprimiti. PL/SQL ima četiri glavne kategorije tipova podatka, a to su:

- skalarni tipovi podataka (*eng. Scalar*) – jedna vrijednost tipa DATE, NUMBER, VARCHAR2, CHAR
- kompozitni tipovi podataka (*eng. Composite*) – podatkovni zapisi koji imaju unutarnje komponente koje su skalarnog tipa podataka te kojima se može pristupiti pojedinačno (zapis, kolekcija)
- referencirani tipovi podataka *(eng. Reference*) – pokazivači na podatkovne zapise (kursor varijabla)
- veliki objekti (*eng. Large Object – LOB*) – pokazivači na velike podatkovne objekte koji se spremaju odvojeno od ostalih podatkovnih zapisa. To su objekti kao tekst, slike, video i audio zapisi.

U ovom poglavlju će se obraditi samo skalarni tipovi podataka, dok će ostali biti obrađeni u posebnim poglavljima.

5.1. Skalarni tipovi podataka

Skalarni tipovi podataka pohranjuju jednu vrijednost.. U kategoriju skalarnih tipova podataka pripadaju tipovi podataka:

- brojčani tip podataka (*eng. numeric data types*)
- znakovni tip podataka (*eng. character data types*)
- BOOLEAN tip podataka
- datumski i vremenski tip podataka (*eng. datetime data types*)
- intervalni tip podataka (*eng. interval data types*)

[9] Oracle (2009) - Oracle Database PL/SQL Language Reference 11g Release 1, *PL/SQL Data Types*, URL: http://docs.oracle.com/cd/B28359_01/appdev.111/b28370/datatypes.htm#i10924, dostupno 14.06.2014.

5.1.1. Brojčani tipovi podataka[10]

Brojčani tipovi podataka (*eng. numeric*) zapisuju brojčane vrijednosti nad kojima se mogu izvoditi aritmetičke operacije.

PLS_INTEGER i BINARY_INTEGER su identični tipovi podataka te spremaju cijele brojeve s predznakom u rasponu od -2 147 483 648 do 2 147 483 647. Sam zapis se sprema pomoću 32 bita, te ovdje ima prednost nad NUMBER tipom podataka koji zauzima više mjesta za pohranu. Druga prednost je da se operacije nad PLS_INTEGER tipom podataka izvode strojno.

BINARY_FLOAT i BINARY_DOUBLE tipovi podataka predstavljaju *single-precision* i *double-precision* zapis broja prema IEEE 754 standardu za zapis decimalnih brojeva. Razlika je u tome što BINARY_FLOAT kao *single_precision* zauzima 32 bita za zapis, što je duplo manje od 64 koje zauzima BINARY_DOUBLE kao *double-precision.*

NUMBER tip podataka može spremati decimalni i cijeli broj. Definira se kao NUMBER(*preciznost, skala*) gdje *preciznost (eng. precision)* predstavlja ukupan broj znamenki, a *skala (eng. scale)* predstavlja broj znamenki nakon decimalne točke. Da bi se definirao cijeli broj koristi se NUMBER(3), gdje broj ne može imati više od 3 znamenke. Decimalni zapis se može zapisati s pomičnom ili fiksnom decimalnom točkom. Za pomičnu decimalnu točku je dovoljno napisati NUMBER, te se ona može zapisati na bilo kojem mjestu. Dok za fiksnu decimalnu točku treba napisati NUMBER(5,2), te sada broj ne može imati više od 5 znamenki i mora imati dvije znamenke nakon decimalne točke. Parametar *preciznost* ima najveću vrijednost 38, dok za parametar *skala* vrijednosti se kreću od -84 i 127. Ako je *skala* podešena na 2, tada će se vrijednost 2.111 zaokružiti na najbližu deseticu odnosno 2.11. A ko je *skala* podešena na -2, tada se vrijednost 2111 zaokružuje na najbližu deseticu s lijeve strane decimalne točke odnosno 2110. Ako je *skala* na svojoj zadanoj vrijednosti koja je 0, tada se decimalni brojevi zaokružuju na najbliži cijeli broj, odnosno vrijednost 2.11 na 2.

[10] Oracle (2009) - Oracle Database PL/SQL Language Reference 11g Release 1, *Predefined PL/SQL Numeric Data Types and Subtypes*, URL: http://docs.oracle.com/cd/B28359_01/appdev.111/b28370/datatypes.htm#i10726, dostupno 14.06.2014.

5.1.2. Znakovni tipovi podataka[11]

Znakovni tipovi podataka (*eng. character*) mogu zapisivati alfanumeričke vrijednosti koje predstavljaju jedan znak ili niz znakova (*eng. string*) sa kojima možemo manipulirati (spajati, rastavljati, pretraživati i slično).

CHAR i NCHAR zapis znakovnog niza fiksne duljine, te se definiraju kao CHAR(*veličina*) i NCHAR(*veličina*). Najveća dopuštena duljina je 32 767 bajtova. Parametar *veličina* (*eng. size*) definira broj znakova koji se sprema u tu varijablu. Ako je broj znakova manji od zadane veličine, tada se razlika broja znakova popuni praznim mjestima. Razlika između CHAR i NCHAR je u standardu zapisa (*eng. charset*) koji se koristi, tako CHAR koristi standard zapisa kakav koristi i sama baza dok NCHAR koristi Unicode, odnosno nacionalni standard zapisa.

VARCHAR2 i NVARCHAR2 zapis znakovnog niza varijabilne duljine, te se definiraju kao VARCHAR(*veličina*) i NVARCHAR(*veličina*). Najveća dopuštena duljina je također 32 767 bajtova. Kod ovog tipa podataka, parametar *veličina* (*eng. size*) definira najveći broj znakova koji se spremaju u tu varijablu. Tako da se sprema samo onoliko znakova koliko je upisano, pod uvjetom da ne prelaze definiranu granicu. Razlika između VARCHAR i NVARCHAR je također u standardu zapisa koji se koristi.

LONG je zapis kao i VARCHAR2 samo što je najveća dopuštena duljina 2 gigabajta, pa može spremati tekst, nizove znakova te čak i kraće dokumente.

LONG RAW je zapis najveće dopuštene duljine 2 gigabajta samo što sprema sirove binarne podatke.

ROWID je oznaka dodatne kolone koja sprema binarne vrijednosti *rowid* koje jednoznačno identificiraju redove te omogućavaju brži pristup retcima. Pomoću funkcije ROWIDTOCHAR se vrijednost pretvara i zapisuje kao CHAR.

```
select  ROWIDTOCHAR(ROWID) into karakter from servis where id_servisa = 1;
```

Gornja naredba daje vrijednost *rowid-a* AAAFJPAABAAAK/xAAA, pomoću kojega se jednoznačno može pristupiti tome retku.

[11] Oracle (2009) - Oracle Database PL/SQL Language Reference 11g Release 1, *Predefined PL/SQL Character Data Types and Subtypes*, URL: http://docs.oracle.com/cd/B28359_01/appdev.111/b28370/datatypes.htm#CIHGBBIG, dostupno 14.06.2014.

5.1.3. BOOLEAN tip podataka[12]

BOOLEAN tip podataka sprema logičke vrijednosti TRUE i FALSE, ali i vrijednost NULL. Vrijednosti tog tipa podataka se mogu koristiti u logičkim operacijama. Ali taj tip podataka se može koristiti samo unutar PL/SQL-a, a ne unutar samog SQL-a. Što znači da se ne koristi u SQL naredbama, predefiniranim SQL funkcijama ni PL/SQL funkcijama koje se pozivaju u SQL naredbama.

U tablicu se ne može unijeti logička vrijednost koja predstavlja TRUE i FALSE, pa se najčešće koriste vrijednost (0,1), ('Y','N'),('true','false') te se kasnije može koristiti IF-THEN ili CASE za prijevod značenja tih stupaca tablice.

5.1.4. Tipovi podataka za označavanje datuma, vremena i intervala[13]

Tipovi podataka omogućavaju pohranu i rad s datumima, vremenima i intervalima, odnosno vremenskim razdobljima. Tri su glavna tipa podataka, a to su DATE, TIMESTAMP i INTERVAL.

DATE predstavlja datumski tip podataka koji sprema datum u formatu dd.MM.yyyy. zajedno s vremenom dana, koje se inače ne prikazuje. Funkcija SYSDATE vraća trenutni datum i vrijeme. Kako bi se maknuo vremenski dio datuma koristi se funkcija TRUNC(SYSDATE) koja vraća datum i vrijeme postavljeno na 00:00:00. Ako želimo saznati vremenski dio DATE tipa podataka možemo koristiti funkciju TO_CHAR(datum, 'hh24:mi:ss') koja vrati samo vrijeme datuma.

```
DECLARE
    datum DATE := SYSDATE;
BEGIN
    DBMS_OUTPUT.PUT_LINE('Datum s vremenom: ' || datum);
    DBMS_OUTPUT.PUT_LINE('Vremenski dio: ' || to_char(datum, 'hh24:mi:ss'));
    DBMS_OUTPUT.PUT_LINE('Datum bez vremena: ' || trunc(datum));
    DBMS_OUTPUT.PUT_LINE('Vremenski dio: ' || to_char(trunc(datum),
'hh24:mi:ss'));
END;

Datum s vremenom: 20.05.14
Vremenski dio: 23:48:10
Datum bez vremena: 20.05.14
Vremenski dio: 00:00:00
```

[12]Oracle (2009) - Oracle Database PL/SQL Language Reference 11g Release 1, *Predefined PL/SQL BOOLEAN Data Type*, URL: http://docs.oracle.com/cd/B28359_01/appdev.111/b28370/datatypes.htm#CJACJGBG, dostupno 14.06.2014.

[13] Oracle (2009) - Oracle Database PL/SQL Language Reference 11g Release 1, *Predefined PL/SQL Datetime and Interval Data Types*, URL: http://docs.oracle.com/cd/B28359_01/appdev.111/b28370/datatypes.htm#i45907, dostupno 14.06.2014.

U varijabli datum je spremljen trenutni datum, te se prvo ispisuje kao što je i deklariran te se dobiva datumski dio. Da se dobije vremenski dio mora se pretvoriti prema nekom predefiniranom uzorku te se za to koristi *'hh24:mi:ss'* format s kojim se dobiva vremenski dio u dvadeset i četiri satnom formatu. Još se može koristiti i format *'hh:mi:ss'* gdje vrijeme nije u dvadeset i četiri satnom formatu. Datum bez vremenskog dijela se ispisuje isto kao i i datum s vremenskim dijelom, ali kada se pokuša ispisati vrijeme datuma bez vremenskog dijela za vrijednost vremena se dobije 00:00:00.

TMESTAMP proširuje DATE format podataka te sprema godinu, mjesec, sat, minutu i sekundu, te se definira kao TIMESTAMP(*preciznost*) gdje je *preciznost* (*eng. precision*) duljinu broja koji se zapisuje poslije sekundi i predstavlja djelić sekunde koji je protekao u zadanoj preciznosti.

TIMESTAMP WITH TIME ZONE je tip podataka kao i TIMEZONE, samo što se na kraju ispisuje vremenska zona.

TIMEZONE WITH LOCAL TIME ZONE isti tip podataka kao i TIMESTAMP WITH TIME ZONE samo što je kada se upisuje vrijednost u bazu, ona se pretvara u vrijednost prema vremenskoj zoni baze podataka te se razlika ne pohranjuje u stupac. A kada se dohvaća iz baze vrijednost se mjenja prema korisničkoj lokalnoj vremenskoj zoni.

```
DECLARE
        vrijeme TIMESTAMP(4) := SYSTIMESTAMP;
        vrijeme_tz TIMESTAMP(4) WITH TIME ZONE := SYSTIMESTAMP;
        vrijeme_tz_loc TIMESTAMP(4) WITH LOCAL TIME ZONE := SYSTIMESTAMP;
BEGIN
        DBMS_OUTPUT.PUT_LINE('Vrijeme: ' || vrijeme);
        DBMS_OUTPUT.PUT_LINE('Vrijeme s vremenskom zonom: ' || vrijeme_tz);
        DBMS_OUTPUT.PUT_LINE('Vrijeme s lokalnom vremenskom zonom: ' ||
        vrijeme_tz_loc);
END;

Vrijeme: 21.05.14 00:22:33,8630
Vrijeme s vremenskom zonom: 21.05.14 00:22:33,8630 +02:00
Vrijeme s lokalnom vremenskom zonom: 21.05.14 00:22:33,8680
```

INTERVAL YEAR TO MONTH je tip podataka koji sprema interval koji predstavlja razdoblje godine i mjeseca. Definira se kao INTERVAL YEAR [(*preciznost*)] TO MONTH gdje je *preciznost* (*eng. precision*) neobavezan podatak koji predstavlja preciznost godina te može biti od 0 do 4, a ako se ne navede zadana vrijednost je 2.

```
DECLARE
        trenutak TIMESTAMP(4) := SYSTIMESTAMP;
        razdoblje INTERVAL YEAR(1) TO MONTH;
```

```
BEGIN
        razdoblje := INTERVAL '5-4' YEAR TO MONTH;
        DBMS_OUTPUT.PUT_LINE('Razdoblje: ' ||razdoblje);

        razdoblje := INTERVAL '9' YEAR;
        DBMS_OUTPUT.PUT_LINE('Razdoblje: ' ||razdoblje);
        razdoblje := INTERVAL '3' MONTH;
        DBMS_OUTPUT.PUT_LINE('Razdoblje: ' ||razdoblje);

        razdoblje := '2-7';
        DBMS_OUTPUT.PUT_LINE('Razdoblje: ' ||razdoblje);

        DBMS_OUTPUT.PUT_LINE('Trenutak: ' || trenutak);
        trenutak := trenutak + razdoblje;
        DBMS_OUTPUT.PUT_LINE('Trenutak + razdoblje: ' || trenutak);
END;

Razdoblje: +5-04
Razdoblje: +9-00
Razdoblje: +0-03
Razdoblje: +2-07
Trenutak: 21.05.14 23:22:04,8590
Trenutak + razdoblje: 21.12.16 23:22:04,8590
```

Postoji više načina kako pridjeliti vrijednost varijabli tog tipa podatka. Pa je tako moguće dodati posebno svaki dio, odnosno posebno godinu i posebno mjesec razdoblja. Moguće je pridjeliti vrijednost kao karakter koji se implicitno pretvara interval. I kao literal, odnosno fiksno zadana vrijednost INTERVAL '5-4' YEAR TO MONTH. Svi primjeri su pokazani u primjeru iznad, te je zadnja vrijednost intervala '2-07' dodana na trenutak u vremenu koji je sada uvećan za dvije godine i sedam mjeseci.

INTERVAL DAY TO SECOND je tip podataka koji sprema interval koji predstavlja broj dana, sati, minuta i sekundi. Sama sintaksa za njegovu deklaraciju je INTERVAL DAY [(*preciznost_dana*)] TO SECOND [(*preciznost_sekundi*)], gdje se odabire preciznost za dan i za dio sekunde, ali ako se ne definira zadane vrijednosti su 2 za dan i 6 za dio sekunde.

```
DECLARE
        trenutak TIMESTAMP(4) := SYSTIMESTAMP;
        razdoblje INTERVAL DAY(2) TO SECOND(4);
BEGIN
          razdoblje := INTERVAL '8 12:30:45' DAY TO SECOND;
          DBMS_OUTPUT.PUT_LINE('Razdoblje: ' ||razdoblje);

          razdoblje := razdoblje + INTERVAL '4' HOUR;
          DBMS_OUTPUT.PUT_LINE('Razdoblje: ' ||razdoblje);

          razdoblje := razdoblje + '1 00:00:00';
          DBMS_OUTPUT.PUT_LINE('Razdoblje: ' ||razdoblje);
END;
```

Razdoblje: +08 12:30:45.0000
Razdoblje: +08 16:30:45.0000
Razdoblje: +09 16:30:45.0000

Kao i za prethodni tip podataka, moguće je u interval dodati svaku vrijednost posebno, kao literal odnosno fiksno te kao karakter koji se implicitno pretvara u interval.

5.2. Usidreni tipovi podataka[14]

Varijabli je moguće dodati i tip podataka koji predstavlja jedan redak u tablici ili stupac u tablici. Tako je sintaksa za deklariranje varijable prema stupcu ili retku sljedeća:

```
naziv_varijable ime_tablice.ime_stupca%TYPE
naziv_varijable ime_tablice%ROWTYPE
```

Prednost je da se kod promjene tipa podataka u tablici ne treba mjenjati PL/SQL kod jer se tip uzima direktno iz tablice što olakšava održavanje baze podataka, odnosno potprograma, okidača i slično.

```
DECLARE
        ime zaposlenik.ime_zap%TYPE;
         id_serv servis.id_servisa%TYPE;
         grad servis.grad_servisa%TYPE;
        red servis%ROWTYPE;
BEGIN
        SELECT z.ime_zap, s.id_servisa, s.grad_servisa
        INTO ime, id_serv, grad
        FROM zaposlenik z
        JOIN servis s on s.id_servisa = z.servis_id_servisa
        WHERE z.id_zap = 1;
        DBMS_OUTPUT.PUT_LINE('Zaposlenik ' || ime ||' radi u servisu ' || grad);

        SELECT * INTO red
        FROM servis WHERE id_servisa = id_serv;
        DBMS_OUTPUT.PUT_LINE('Servis se nalazi na adresi: ' || red.adresa_servisa);
END;

Zaposlenik Ivica radi u servisu Zagreb
Servis se nalazi na adresi: Ilica 58
```

U primjeru iznad definirane su tri varijable kojima je za tip podataka pridružen tip podataka stupca u određenoj tablici, te je definirana još jedna varijabla kojoj je tip podataka tip cijelog retka u tablici. Prvim upitom dobavimo ime servisera, identifikator servisa i grad servisa te spremimo podatke u za to predviđene varijable. Koristeći varijablu koja predstavlja identifikator servisa, drugim upitom se dohvaća cijeli redak koji predstavlja taj servis. Za pristup komponentama te varijable, odnosno retka koristi se sintaksa:

[14] Rosenzweig B., Rakhimov E. (2009) – *Oracle PL/SQL by Example Fourth Edition*, str.29.

ime_varijable.ime_stupca

U ovom primjeru prikazan dohvat i korištenje usidrenih tipova podataka u PL/SQL-u. Najvažnija prednost je ta da se uklanja potreba za tipovima podataka te održavanjem PL/SQL koda kod svake promjene na bazi. Primjerice za sada je tip podataka za stupac *adresa_servisa* u tablici *servis* VARCHAR2(30) i ako bi se on u budućnosti povećao na VARCHAR2(45) u njega bi se upisivale adrese koje su do te duljine. U isto vrijeme je PL/SQL kod ostao isti i varijabla ostala pod tipom podataka VARCHAR2(30). Sada kod dohvaćanja varijable koja prelazi duljinu 30 za taj tip podataka doći do greške zbog neprikladnog tipa podataka. Zato usidreni tipovi podataka eliminiraju potrebu za održavanjem tipova podataka u kodu, jer se tipovi uzimaju direktno iz tablice.

6. Operatori[15]

Operatori omogućavaju rad sa varijablama, odnosno operandima te vraćanje rezulatata. Svaki operand ima rezervirani znak ili ključnu riječ koja se pojavljuje ispred operanda ili između dva operanda. To ovisi o tome je li operand unaran ili binaran. Ako je unaran tada se pojavljuje kao '***operator*** *operand'*, dok se binaran pojavljuje kao *'operand* ***operator*** *operand'*. A postoje čak i operatori koji prihvaćaju više operanada.

6.1. Aritmetički operatori

Aritmetički operatori omogućavaju rad s brojevima, odnosno rad s varijablama koje su numeričkog tipa podataka. Operatori su:

- zbrajanje (+) – prvom operandu dodaje drugi operand
- oduzimanje (-) – od prvog operanda oduzima drugi operand
- množenje (*) – množi prvi operand s drugim operandom
- djeljenje (/) – dijeli prvi operand s drugim operandom
- potenciranje (**) – potencira prvi operand na drugi operand

Aritmetički operatori omogućavaju rad s brojevima, odnosno njihovo zbrajanje, oduzimanje, množenje i potenciranje. Koriste se nad numeričkim tipovima podataka.

6.2. Operatori usporedbe

Operatori usporedbe uspoređuju dva izraza. Rezultat je BOOLEAN vrijednost, a operatori su:

- operator jednakosti (=) – uspoređuje jednakost dviju vrijednosti
- operator različitosti (!=, <>, ~=) – uspoređuje različitost dviju varijabli
- veći od (>) – uspoređuje da li je prva vrijednost veća od druge vrijednosti
- manji (<) - uspoređuje da li je prva vrijednost manja od druge vrijednosti
- veći ili jednak (>=) - uspoređuje da li je prva vrijednost veća ili jednaka drugoj
- manji ili jednak (<=) - uspoređuje da li je prva vrijednost manja ili jednaka drugoj
- **LIKE** – operator provjerava da li prvi izraz odgovara drugom izrazu ili dijelu drugog izraza. U izrazima se može koristiti znak % koji označava da dio riječi može biti koja kombinacija znakova i znak _ koji zamjenjuje jedan znak izraza. Primjerice *'Crna*

[15] TutorialsPoint (2014) - PL/SQL Tutorial, *PL/SQL - Operators*, URL: http://www.tutorialspoint.com/plsql/plsql_operators.htm, dostupno 14.06.2014

boja' LIKE '%oja' vraća TRUE jer znak % u ovom slučaju zamjenjuje izraz 'Crna b'. U slučaju da je izraz bio *'Crna boja' LIKE '%xja'* rezultat bi bio FALSE. U izrazu *'boja' LIKE '_oja'* rezultat bi bio TRUE jer znak _ zamjenjuje slovo 'b', dok bi u izrazu *'bxoja' LIKE '_oja'* rezultat bio FALSE.

- **BETWEEN** – operator provjerava da li je neka vrijednost unutar nekog intervala. Operacija se provjerava izrazom *vrijednost BETWEEN pocetna_vrijednost AND krajnja_vrijednost*. Rezultat je istinit odnosno TRUE ako je vrijednost veća ili jednaka početnoj vrijednosti i manja ili jednaka krajnjoj. Za napomenuti je da je važno da početna vrijednost bude manja od prve, jer ako se uspoređuje da li je broj 3 u intervalu od 1 do 5 vratit će se TRUE. Dok će se za interval od 5 do 1 vratiti FALSE.
- **IN** – operator provjerava da li se neka vrijednost nalazi u skupu nekih vrijednosti, te se provjerava izrazom *vrijednost IN (*$vrijednost_1$*, ...,* $vrijednost_N$*)*. Ako se vrijednost nalazi u tom skupu vrijednosti, rezultat je TRUE dok je u suprotnom FALSE.
- **IS NULL** – provjerava da li je neka vrijednost jednaka *null*. U SQL-u se općenito ne koristi izraz jednakosti za provjeru da li je vrijednost jednaka *null* nego operator IS NULL i njegova varijacija IS NOT NULL. Za napomenuti je da CHAR ili VARCHAR2 tip podataka koji imaju samo *whitespaces* nije jednak *null*, ali ako se koristi funkcija TRIM nad tom vrijednosti, ona makne *whitespaces* pa će vrijednost biti jednaka *null* te će se u tom slučaju vratiti TRUE.

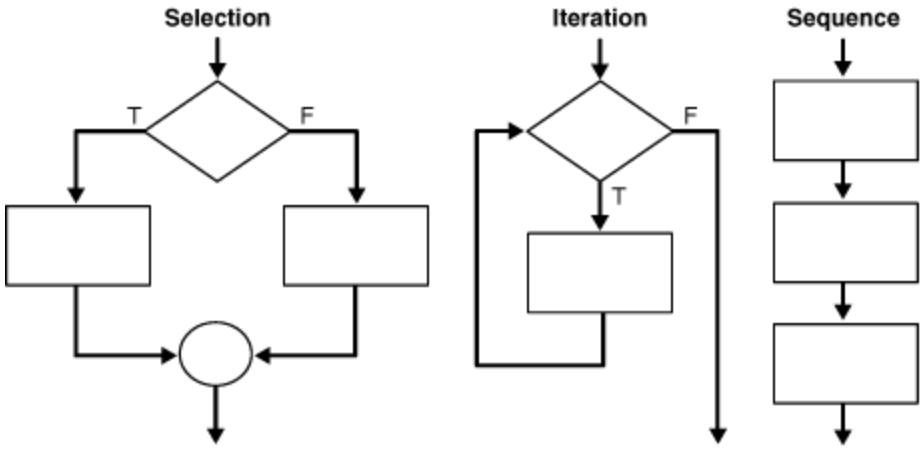
6.3. Logički operatori

Pomoću logičkih operatora grade se složeniji uvjeti koji se ispituju. Rade samo s logičkim, odnosno BOOLEAN izrazima i kao rezultat također daju BOOLEAN izraz.

- AND operator – daje TRUE samo ako su oba izraza istinita, odnosno TRUE
- OR operator – daje TRUE ako je bilo koji od dva izraza istinit, odnosno TRUE
- NOT operator – unarni operator koji mjenja logičku vrijednost iz TRUE u FALSE i obrnuto

7. Izrazi za kontrolu programskog toka PL/SQL-a

Izrazi za kontrolu uključuju selekciju, iteraciju i sekvencu. Oni omogućavaju testiranje uvjeta, petlje te grananje kako bi se mogli napisati dobro strukturirani PL/SQL programi. U programerskom smislu to znači da se naredbe ne moraju izvršavati sekvencijalno, nego da se skupina naredbi izvrši ili ne izvrši ovisno o tome da li je ispunjen određeni upit.[16] Prikaz izvođenja tih kontrolnih struktura je prikazan slikom 4.



Slika 4. Kontrolne strukture[17]

7.1. Selekcija[18]

Selekcija omogućava da ovisno o ispunjenosti uvjeta se izvrši određena grupa naredbi. Ako uvjet nije ispunjen, može se izvršiti druga grupa naredbi. Selekcija se ugrubo može podjeliti na IF i CASE kontrole.

IF-THEN je najjednostavnija inačica selekcije koja provjerava jedan uvjet, te u slučaju da je on istinit odnosno TRUE, izvršava naredbe koje se nalaze između ključnih riječi THEN i END IF. Ako je uvjet FALSE ili NULL ne izvršava se ništa.

```
IF uvjet THEN
        naredbe_koje_se_izvršavaju
END IF;
```

[16] Rosenzweig B., Rakhimov E. (2009) – *Oracle PL/SQL by Example Fourth Edition*, str.53.
[17] Oracle (2009) - Oracle Database PL/SQL Language Reference 11g Release 1, *Overview of PL/SQL Control Structures*, URL: http://docs.oracle.com/cd/B28359_01/appdev.111/b28370/controlstructures.htm #LNPLS00401, dostupno 14.06.2014.
[18] Oracle (2009) - Oracle Database PL/SQL Language Reference 11g Release 1, *Testing Conditions*, URL: http://docs.oracle.com/cd/B28359_01/appdev.111/b28370/controlstructures.htm#LNPLS00402, dostupno 14.06.2014.

IF-THEN-ELSE je dosta slična kao i IF-THEN, samo je razlika u tome što se definiraju i naredbe koje se izvršavaju u slučaju da je uvjet FALSE ili NULL. Naredbe u slučaju istinitosti uvjeta se nalaze između ključnih riječi THEN i ELSE, dok se za drugi slučaj naredbe nalaze između ključnih riječi ELSE i END IF.

```
IF uvjet THEN
        naredbe_uvjet_ispunjen
ELSE
        naredbe_uvjet_neispunjen
END IF;
```

IF-THEN-ELSIF omogućava testiranje više uvjeta te odabir naredbi za izvođenje na temelju tih uvjeta. Prvi uvjet se testira sa IF, dok svaki svaki sljedeći se testira s ključnom riječi ELSIF. Na kraju još postoji i ELSE ključna riječ nakon koje se nalaze naredbe koje se izvršavaju ako se nije izvršio ni jedan dio iznad.

```
IF uvjet_1 THEN
        naredbe_uvjet_1_ispunjen
ELSIF uvjet_2 THEN
        naredbe_uvjet_2_ispunjen
ELSE
        naredbe_uvjeti_neispunjeni
END IF;
```

Sve IF selekcije se mogu međusobno ugnježđivati pa je u sljedećem primjeru prikazano ugnježđivanje sa IF-THEN, IF-THEN-ELSE i IF-THEN-ELSIF naredbama. Unutar IF-THEN se ispituje da li je varijabla jednaka *null*. Ako nije izvršava se unutarnji dio koji je IF-THEN-ELSE naredba koja ispituje da li je ocjena pozitivna ili negativna te ovisno o rezultatu izvršava jedan od dva dijela. Ako je ocjena pozitivna, pomoću IF-THEN-ELSIF naredbe se ispituje koja je ocjena te ovisno o tome dodjeljuje vrijednosti u drugu varijablu. Tako se mogu izvršavati različite grupe naredbi ovisno o uvjetima.

```
DECLARE
        ocjena INTEGER := 4;
        naziv VARCHAR2(10);
BEGIN
        IF ocjena IS NOT NULL THEN
                IF ocjena >= 2 AND ocjena <= 5 THEN
                        IF ocjena = 2 THEN
                                naziv := 'Dovoljan';
                        ELSIF ocjena = 3 THEN
                                naziv := 'Dobar';
                        ELSIF ocjena = 4 THEN
                                naziv := 'Vrlo dobar';
                        ELSIF ocjena = 5 THEN
                                naziv := 'Izvrstan';
                        END IF;
```

```
                DBMS_OUTPUT.PUT_LINE('Prolaz na ispitu, ocjenom ' || naziv);
            ELSE
                DBMS_OUTPUT.PUT_LINE('Pad na ispitu');
            END IF;
        END IF;
END;
```

Prolaz na ispitu, ocjenom Vrlo dobar

CASE je veoma sličan IF-THEN-ELSE naredbi, samo je razlika da umjesto više uvjeta kojima se ispituje istinitost, koristi selektor. Započinje ključnom riječi CASE nakon koje dolazi selektor i nakon toga se navodi više izjava kojima se ispituje selektor. Započinje ključnom riječi WHEN nakon koje dolazi vrijednost, te ako je selektor jednak toj vrijednosti izvršava se blok naredbi koje se nalaze iza ključne riječi THEN i prije sljedeće ključne riječi WHEN ili ELSE. ELSE se izvršava ako ni jedna izjava iznad nije bila izvršena. Sam blok završava ključnom riječi END CASE.

```
CASE selektor
    WHEN 'X' THEN naredbe_selektor_X
    WHEN 'Y' THEN naredbe_selektor_Y
    ELSE naredbe _default
END CASE;
```

Postoji i druga verzija CASE selekcije koja nema selektor nego pretražuje na način da uspoređuje varijablu između ključnih riječi WHEN i THEN, te ako usporedba rezultira istinitim rezultatom izvodi se taj blok izjava.

```
CASE
    WHEN selektor =  'X' THEN naredbe_selektor_X
    WHEN selektor  = 'Y' THEN naredbe_selektor_Y
    ELSE naredbe_default
END CASE;
```

Isti primjer kao i za IF grupu naredbi je prikazan u sljedećem primjeru, te se ovdje koristi CASE naredba sa selektorom. Verzija bez selektora bi se razlikovala samo po tome što bi se varijabla ocjena uspređivala sa vrijednosti između ključnih riječi WHEN i THEN.

```
DECLARE
    ocjena INTEGER := 4;
    naziv VARCHAR2(10);
BEGIN
    IF ocjena IS NOT NULL AND ocjena BETWEEN 2 AND 5 THEN
        CASE ocjena
            WHEN 2 THEN naziv := 'Dovoljan';
            WHEN 3 THEN naziv := 'Dobar';
            WHEN 4 THEN naziv := 'Vrlo dobar';
            WHEN 5 THEN naziv := 'Izvrstan';
        END CASE;
```

```
            DBMS_OUTPUT.PUT_LINE('Prolaz na ispitu, ocjenom ' || naziv);
        ELSE
            DBMS_OUTPUT.PUT_LINE('Pad na ispitu');
        END IF;
END;
```

Prolaz na ispitu, ocjenom Vrlo dobar

7.2. Iteracija[19]

Iteracije se koriste kada se blok naredbi treba izvršiti više puta. Blok naredbi se izvodi sve dok je uvjet za izvođenje iteracije, odnosno petlje ispunjen.

Osnovna petlja okružuje blok naredbi koje se trebaju izvršiti između dvije ključne riječi LOOP i END LOOP.

```
LOOP
    blok_naredbi;
END LOOP;
```

Ova petlja nema ispitivanja uvjeta koji određuje kada se prekida izvođenje petlje. Zato se koriste naredbe EXIT i EXIT WHEN. Naredba EXIT se koristi zajedno sa IF-THEN naredbom, dok se EXIT WHEN može koristiti sama.

```
DECLARE
    x NUMBER := 1;
BEGIN
    LOOP
        x := x + 10;
        IF x > 10 THEN
            EXIT;
        END IF;
    END LOOP;
END;
```

```
DECLARE
    x NUMBER := 1;
BEGIN
    LOOP
        x := x + 10;
        EXIT WHEN x > 10;
    END LOOP;
END;
```

U sljedećem primjeru će se prikazati izvođenje osnovne petlje na jednostavnom primjeru koji ispituje djeljivost broja sa brojevima od 1 do 10. Izlaz iz petlje je ostvaren EXIT WHEN naredbom koja provjerava uvjet, te kada je on ispunjen izlazi iz petlje.

```
DECLARE
    n INTEGER := 85;
    brojac INTEGER := 1;
BEGIN
    LOOP
        IF(MOD(n, brojac) = 0) THEN
            DBMS_OUTPUT.PUT_LINE(n || ' je djeljiv s ' || brojac);
        END IF;
```

[19] Oracle (2009) - Oracle Database PL/SQL Language Reference 11g Release 1, *Controlling Loop Iterations*, URL: http://docs.oracle.com/cd/B28359_01/appdev.111/b28370/controlstructures.htm#LNPLS00403, dostupno 14.06.2014.

```
            brojac := brojac + 1;
            EXIT WHEN brojac > 10;
        END LOOP;
END;

Broj 85 je djeljiv s 1
Broj 85 je djeljiv s 5
```

WHILE petlja je petlja koja se izvršava neodređeni broj puta, odnosno izvršava se sve dok je uvjet istinit.

```
WHILE uvjet LOOP
    blok_naredbi;
END LOOP;
```

Isti primjer kao za osnovnu LOOP petlju je sada napravljen pomoću WHILE petlje. Razlika je u tome što se uvjet mora biti ispunjen definira nakon ključne riječi WHILE, te petlja sama prekida izvođenje kada uvjet više nije ispunjen.

```
DECLARE
    n INTEGER := 85;
    brojac INTEGER := 1;
BEGIN
    WHILE brojac <= 10 LOOP
        IF(MOD(n, brojac) = 0) THEN
            DBMS_OUTPUT.PUT_LINE(n || ' je djeljiv s ' || brojac);
        END IF;
        brojac := brojac + 1;
    END LOOP;
END;

85 je djeljiv s 1
85 je djeljiv s 5
```

FOR petlja je petlja koja se izvršava određeni broj puta, što znači da je broj ponavljanja bloka naredbi unaprijed određen.

```
FOR brojac IN početna_vrijednost ... krajnja_vrijednost LOOP
    blok_naredbi;
END LOOP;
```

Primjer za FOR petlju ima prednost nad WHILE petljom, a to je da se ne mora unutar petlje povećavati brojač, odnosno varijabla o kojoj ovisi uvjet nego se to obavlja direktno unutar petlje. FOR petlja može i brojati unatrag, samo se umjesto ključne riječi IN stavi IN REVERSE. Nedostatak nad WHILE petljom je taj da je promjena, odnosno inkrement uvijek jedan. Dok se u WHILE petlji može definirati za koliko se mijenja varijabla.

```
DECLARE
    n INTEGER := 85;
```

```
BEGIN
        FOR brojac IN 1 .. 10 LOOP
                IF(MOD(n, brojac) = 0) THEN
                        DBMS_OUTPUT.PUT_LINE(n || ' je djeljiv s ' || brojac);
                END IF;
        END LOOP;
END;

85 je djeljiv s 1
85 je djeljiv s 5
```

8. Kursori

Kursori (*eng. cursor*) su memorijski prostori gdje su pohranjene informacije koje su potrebne da Oracle izvršava SQL naredbe.[20] PL/SQL upravlja tim memorijskim prostorom pomoću kursora koji služi kao pokazivač na taj memorijski prostor. U kursoru se nalaze podatci kao broj redaka koji su dohvaćeni upitom, retci koji su dohvaćeni i slično. Retci koji se nalaze u kursoru još se nazivaju aktivan skup (*eng. active set*). Najvažnija karakteristika kursora je da on može biti imenovan kako bi se mogao pozivati u kodu, te omogućava obradu svih redaka koji se nalaze u njemu. Postoje dvije vrste kursora[21]:

- implicitni kursor
- eksplicitni kursor

8.1. Implicitni kursor[22]

Svaki put kada se izvrši SQL naredba, Oracle implicitno kreira kursor. Implicitni kursor se kreira za DML (*eng. data manipulation language*) naredbe, odnosno UPDATE, DELETE i INSERT naredbe. UPDATE i DELETE naredbe imaju kursor koji identificira retke koji će se promijeniti ili obrisati, dok za INSERT naredbu kursor sprema podatke koji se upisuju bazu podataka. Implicitni kursor se koristi još za SELECT INTO naredbu, koja dobavlja samo jedan red te njegove vrijednosti upisuje u neku varijablu. Zadnji otvoreni implicitni kursor se naziva SQL kursor, te mu se pristupa preko oznake *SQL*. Preko te oznake se može saznati koliko je redova bilo zahvaćeno zadnjom SQL naredbom, koristeći %ROWCOUNT nad SQL oznakom, odnosno SQL%ROWCOUNT.

```
DECLARE
        broj INTEGER;
BEGIN
        UPDATE nalog SET cijena = 0;
        broj := SQL%ROWCOUNT;
        DBMS_OUTPUT.PUT_LINE('Broj redaka: ' || broj);

        UPDATE nalog SET cijena = 1 WHERE idnalog = 54;
        broj := SQL%ROWCOUNT;
        DBMS_OUTPUT.PUT_LINE('Broj redaka: ' || broj);
END;

Broj redaka: 3134
Broj redaka: 1
```

[20] Rosenzweig B., Rakhimov E. (2009) – *Oracle PL/SQL by Example Fourth Edition*, str.229.
[21] Rosenzweig B., Rakhimov E. (2009) – *Oracle PL/SQL by Example Fourth Edition*, str.230.
[22] Rosenzweig B., Rakhimov E. (2009) – *Oracle PL/SQL by Example Fourth Edition*, str.231.

8.2. Eksplicitni kursor[23]

Za razliku od implicitnog kursora kojeg deklarira Oracle, eksplicitni kursor treba deklarirati u programu. On se koristi za upite koji vraćaju više od jednog retka, te se deklarira u deklarativnom dijelu PL/SQL bloka. Nakon toga se u izvršnom dijelu bloka mogu obrađivati retci koji se nalaze u tom kursoru. Rad s eksplicitnim kursorom se sastoji od četiri koraka:

- deklariranje kursora (*eng. declaring*) – deklaracija kursora u memoriji
- otvaranje kursora (*eng. opening)* – otvaranje deklariranog kursora i dodjeljivanje memorije
- dohvaćanje kursora (*eng. fetching*) – dohvaćanje podataka iz deklariranog i otvorenog kursora
- zatvaranje kursora (*eng. closing*) – deklarirani, otvoreni i dohvaćeni kursor se zatvara da se alocirana memorija oslobodi

Kursor se deklarira u deklarativnom dijelu PL/SQL bloka gdje mu se dodijeli ime za koje je uobičajena praksa da započne sa *c_* kako bi se dalje u bloku znalo da se radi o kursoru. Uz to još mu se pridruži SELECT naredba, te je sintaksa za deklariranje sljedeća:

CURSOR ime_kursora IS select_naredba;

Nakon deklariranja slijedi otvaranje kursora kod kojega se provjeri WHERE klauzula SELECT naredbe, odredi aktivan skup, odnosno retci koji se zapisuju u kursor te se pokazivač aktivnog skupa postavlja na prvi redak. Sintaksa za otvaranje kursora je:

OPEN ime_kursora;

Kada je kursor deklariran i otvoren, podatci iz njega se mogu dohvatiti. Iz kursora se dohvaća red na koji pokazuje pokazivač kursora, te se podatci iz reda zapisuju u varijable deklariranu u deklarativnom dijelu bloka. Sintaksa za dohvaćanje retka iz kursora je:

FETCH ime_kursora INTO varijable

Kursor se može zapisati u više varijabli ili ako se dobavljaju podatci iz jedne tablice uobičajeno je definirati varijablu koja je tipa %ROWTYPE. Nakon dohvaćanja jednog retka, pokazivač kursora se pomiče na sljedeći red aktivnog skupa. Taj postupak se ponavlja sve dok se ne prođu svi retci u aktivnom skupu. Zbog toga je uobičajeno koristiti osnovnu petlju gdje se uvjet izlaza kontrolira pomoću %NOTFOUND nad kursorom.

[23] Rosenzweig B., Rakhimov E. (2009) – *Oracle PL/SQL by Example Fourth Edition*, str.232.

```
DECLARE
        CURSOR c_servis IS SELECT * FROM servis;
        grad servis%ROWTYPE;
BEGIN
        OPEN c_servis;
        LOOP
                FETCH c_servis INTO grad;
                EXIT WHEN c_servis%NOTFOUND;
                DBMS_OUTPUT.PUT_LINE(grad.adresa_servisa || ', ' || grad.grad_servisa);
        END LOOP;
        CLOSE c_servis;
END;

Ilica 58, Zagreb
Radnička cesta 12, Zagreb
Ulica braće Radić 2, Varaždin
```

Kada se obrade svi retci iz kursora, on se treba zatvoriti kako bi se oslobodila alocirana memorija. U bloku je moguće imati više otvorenih kursora, te se kod kompliciranijih blokova može zauzeti puno memorije ako se ne vodi računa o njihovom zatvaranju. Sintaksa za zatvaranje kursora je

```
CLOSE ime_kursora;
```

Atributi koji se koriste kako bi se odredile neke karakteristike kursora kada je on otvoren ili dohvaćen:

- *ime_kursora%NOTFOUND* – BOOLEAN vrijednost, koja je TRUE ako prethodni FETCH nije vratio red
- *ime_kursora%FOUND* – BOOLEAN vrijednost, koja je TRUE ako je prethodni FETCH vratio red
- *ime_kursora%ROWCOUNT* – broj redaka u kursoru
- *ime_kursora%ISOPEN* – BOOLEAN vrijednost, koja je TRUE ako je kursor otvoren

Postoji i elegantniji način za rad sa kursorima, a to je pomoću FOR petlje koja pojednostavljuje rad. Otvaranje, dohvaćanje i zatvaranje kursora se radi implicitno te je lakše pisati i održavati kod.

```
DECLARE
        CURSOR c_servis IS SELECT * FROM servis;
BEGIN
        FOR r IN c_servis LOOP
                DBMS_OUTPUT.PUT_LINE(r.adresa_servisa || ', ' || r.grad_servisa);
        END LOOP;
END;
```

Ilica 58, Zagreb
Radnička cesta 12, Zagreb
Ulica braće Radić 2, Varaždin

U primjeru iznad je pokazan isti kod kao sa osnovnom LOOP petljom, te je kod puno *čišći* zbog toga jer FOR petlja implicitno prepoznaje kursor te obavlja sve korake potrebne za rad sa njim.

8.2.1. Kursori s parametrima[24]

Kursoru se kod deklaracije mogu dodati i parametri, kako bi se isti mogao ponovno koristiti i dati druge rezultate, odnosno aktivan skup. Kursoru se vrijednost parametra proslijeđuje kod njegovog otvaranja, ali mu se može dodjeliti i DEFAULT vrijednost koja se koristi ako nije ništa proslijeđeno. Vrijednost samog parametra je vidljiva samo unutar kursora.

```
DECLARE
        CURSOR c_zaposlenici (p_servis IN servis.id_servisa%TYPE DEFAULT 2) IS
                SELECT z.ime_zap, z.prezime_zap, s.grad_servisa
                FROM zaposlenik z
                JOIN servis s on s.id_servisa = z.servis_id_servisa
                WHERE s.id_servisa = p_servis;
BEGIN
        FOR r IN c_zaposlenici(1)
        LOOP
                DBMS_OUTPUT.PUT_LINE(r.ime_zap || ' ' ||r.prezime_zap || ', ' ||
        r.grad_servisa);
        END LOOP;
END;

Ivica Ivic, Zagreb
Marija Maric, Zagreb
Marko Markic, Zagreb
```

Gornji primjer prikazuje kursor koji ima DEFAULT vrijednost 2, te prima parametar koji je istog tipa kao i stupac *id_servisa* u tablici *servis*. U izvršnom dijelu mu se proslijeđuje identifikator servisa, odnosno parametar vrijednosti 1 te kursor vrati zaposlenike koji rade u tom servisu. Da kursoru nije proslijeđen parametar vratio bi sve zaposlenike koji rade u servisu koji ima identifikator 2. U slučaju da kursor nema DEFAULT vrijednost, dolazi do greške ako mu se ne proslijedi parametar.

[24] Rosenzweig B., Rakhimov E. (2009) – *Oracle PL/SQL by Example Fourth Edition*, str.254.

8.2.2. Ugnježđeni kursori[25]

U PL/SQL-u kursori se mogu i ugnježđivati, te se tako vrijednosti koje vraća jedan kursor mogu proslijediti drugom kursoru koji vraća retke prema tim vrijednostima.

```
DECLARE
        CURSOR c_servisi IS
                SELECT id_servisa, adresa_servisa, grad_servisa
                FROM servis;
        CURSOR c_zaposlenici (p_servis IN servis.id_servisa%TYPE) IS
                SELECT ime_zap, prezime_zap
                FROM zaposlenik
                WHERE servis_id_servisa = p_servis;
BEGIN
        FOR s IN c_servisi
        LOOP
                DBMS_OUTPUT.PUT_LINE('SERVIS: ' || s.adresa_servisa || ', ' ||
                s.grad_servisa);
                FOR r IN c_zaposlenici(s.id_servisa)
                LOOP
                        DBMS_OUTPUT.PUT_LINE(r.ime_zap || ' ' ||r.prezime_zap);
                END LOOP;
        END LOOP;
END;
```

```
SERVIS: Ilica 58, Zagreb
Ivica Ivic
Marija Maric
Marko Markic
SERVIS: Radnička cesta 12, Zagreb
Kruno Krunic
Matija Matic
Kreso Kresic
SERVIS: Ulica braće Radić 2, Varaždin
Petar Petric
Slavko Slavic
Sanja Sanjic
```

U primjeru iznad je prikazano ugnježđivanje kursora gdje prvi kursor prolazi kroz sve servise, te za svaki dobiveni servis u prvom kursoru se dohvaćaju retci, odnosno zaposlenici tog servisa u drugom kursoru.

[25] Rosenzweig B., Rakhimov E. (2009) – *Oracle PL/SQL by Example Fourth Edition*, str.255.

8.2.3. UPDATE i DELETE redaka aktivnog seta kursora[26]

Do sada su obrađeni kursori koji samo dohvaćaju retke, ali postoje još kursori koji se koriste za promjenu ili brisanje redaka koji se nalaze u aktivnom setu kursora. Ako se će se kursor koristiti za promjenu ili brisanje koristi se sljedeća sintaksa[27]:

```
CURSOR ime_kursora
IS
        select_naredba
        FOR UPDATE [OF imena_stupaca] [NOWAIT];
```

Na kraju kursora se dodaje FOR UPDATE ključna riječ koja zaključava retke na koje se odnosi kursor, što znači da ostali korisnici ne mogu mijenjati te retke prije nego se izvrši promjena ili retci ne otključaju. Retci se otključavaju korištenjem ključne riječi COMMIT ili ROLLBACK. Opcionalni dijelovi su još ključna riječ OF iza koje se navode imena stupaca na kojima će se izvoditi promjene. Ako se koristi više tablica unutar SELECT naredbe kursora, može se navesti redak iz jedne koji se namjerava mjenjati te će se tada zaključati samo ta tablica dok će preostale tablice biti na raspolaganju drugim korisnicima. Ako se pokuša pristupiti retcima koji su već zaključani Oracle inače čeka dok se ti retci ne oslobode. Ali ako se navede ključna riječ NOWAIT, kontrola se vraća programu koji se izvodi kako bi nastavio raditi prije nego opet proba pristupiti retcima.[28]

```
DECLARE
        CURSOR c_zaposlenici (p_servis IN servis.id_servisa%TYPE) IS
                SELECT *
                FROM zaposlenik
                WHERE servis_id_servisa = p_servis
                FOR UPDATE OF placa_zap NOWAIT;
BEGIN
        FOR r IN c_zaposlenici(2)
        LOOP
                UPDATE zaposlenik
                SET placa_zap = r.placa_zap + 100
                WHERE CURRENT OF c_zaposlenici;
        END LOOP;
        COMMIT;
END;
```

U primjeru iznad je korištena klauzula koja se koristi kada se mijenjaju ili brišu retci koji se nalaze u aktivnom setu kursora, a to je WHERE CURRENT OF koja pokazuje zadnje

[26] Rosenzweig B., Rakhimov E. (2009) – *Oracle PL/SQL by Example Fourth Edition*, str.258.
[27] Tech On The Net (2014) - Oracle/ PLSQL, *SELECT FOR UPDATE Statement*, URL: http://www.techonthenet.com/oracle/cursors/for_update.php, dostupno 14.06.2014.
[28] Oracle (2005) - Oracle Database PL/SQL Language Reference 10g Release 2, *Using FOR UPDATE*, URL: http://docs.oracle.com/cd/B19306_01/appdev.102/b14261/sqloperations.htm#i3314, dostupno 14.06.2014.

dohvaćeni redak kursora.[29] Nakon izvršene promjene, odnosno povećanja plaće zaposlenicima servisa pod identifikatorom 2, promjene se predaju ključnom riječi COMMIT, nakon koje se retci otključavaju. Može se i cijela transakcija poništiti ključnom riječi ROLLBACK, te da je ona umjesto COMMIT u ovom primjeru promjena se ne bi zapisala. Kod velikog broja redaka koji se mjenjaju, programer može definirati da se COMMIT izvršava svakih nekoliko tisuća redaka, tako da u slučaju greške ROLLBACK nije prevelik.

[29] Tech On The Net (2014) - Oracle/ PLSQL, *WHERE CURRENT OF Statement*, URL: http://www.techonthenet.com/oracle/cursors/current_of.php, dostupno 14.06.2014.

9. Kolekcije[30]

Kolekcije u PL/SQL-u omogućavaju zapisivanje grupe elemenata istog tipa podataka, te funkcioniraju kao polja u ostalim programskim jezicima. Svakom elementu kolekcije se pristupa pomoću indeksa, koji u PL/SQL-u započinju od broja 1 dok je u ostalim programskim jezicima uobičajeno da indeksi započinju od broja 0. Kolekcije u PL/SQL-u se djele na:

- Asocijativni niz (*eng. Associative Arrays*) (još poznato kao *Index-By Tables*)
- Ugnježđene tablice (*eng. Nested Tables*)
- Polja varijabilne duljine (*eng. Variable-Size Arrays*) (još poznato kao *Varrays*)

9.1. Asocijativni niz[31]

Asocijativni niz je kolekcija koja se sastoji od ključa i vrijednosti, slično kao *hash* tablice u ostalim programskim jezicima. Definira se sljedećom sintaksom:

```
TYPE ime_tipa IS TABLE OF tip_elementa [NOT NULL]
        INDEX BY tip_elementa;
ime_tablice IME_TIPA;
```

Deklaracija se sastoji od dva koraka, gdje se u prvom koraku definira tip tablice. Odabere se ime koje će se koristiti za taj tip i koje će se koristiti u drugom koraku kao tip podataka. Nakon TABLE OF ključne riječi dolazi tip podataka za polja, odnosno vrijednosti koje će se upisivati te se uz to može definirati da li vrijednost može biti *null*. Nakon ključne riječi INDEX BY se odabire tip podataka za ključ, odnosno tip podataka po kojem će se pretraživati kolekcija i dobavljati vrijednosti. Može se pretraživati po brojevima ili po znakovima. Drugi korak je kao normalno deklariranje varijable, osim što se za tip podataka koristi ime tipa deklarirano u prvom koraku.

```
DECLARE
        CURSOR c_servis IS
                SELECT * FROM servis;
        TYPE servisi IS TABLE OF VARCHAR2(50)
                INDEX BY BINARY_INTEGER;
        tablica servisi;
BEGIN
        FOR r IN c_servis LOOP
                tablica(r.id_servisa) := r.adresa_servisa || ', ' || r.grad_servisa;
                DBMS_OUTPUT.PUT_LINE('Vrijednost: ' || tablica(r.id_servisa));
        END LOOP;
END;
```

[30] Rosenzweig B., Rakhimov E. (2009) – *Oracle PL/SQL by Example Fourth Edition*, str.315.
[31] Rosenzweig B., Rakhimov E. (2009) – *Oracle PL/SQL by Example Fourth Edition*, str.317.

Vrijednost: Ilica 58, Zagreb
Vrijednost: Radnička cesta 12, Zagreb
Vrijednost: Ulica braće Radić 2, Varaždin

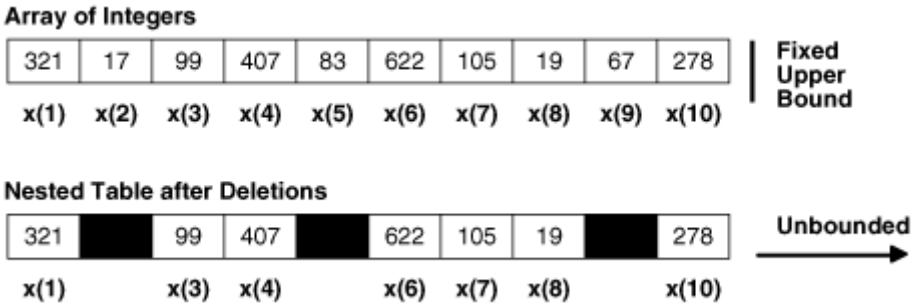
U primjeru iznad je kreiran asocijativni niz pod imenom *servisi*, te kasnije varijabla *tablica* koja ima taj tip. U tu varijablu su spremljene lokacije servisa dohvaćene kursorom. Lokacije su spremljene kao vrijednosti, dok im je ključ identifikator koji se koristi u tablici. Odnosno za red u tablici pod identifikatorom 2, odgovarajuću vrijednost u kolekciji nalazimo tako da ju pretražujemo za ključ 2 te dobivamo odgovarajuću lokaciju.

9.2. Ugnježđene tablice[32]

Ugnježđena tablica je kolekcija koja odgovara jednodimenzionalnom polju sa proizvoljnim brojem elemenata. Definira se dva koraka, slično kao i asocijativni niz samo bez INDEX BY ključne riječi jer će se elementima pristupati pomoću indeksa koji je može biti samo cjelobrojna vrijednost. Sama definicija je dana sintaksom:

*TYPE **ime_tipa** IS TABLE OF tip_elementa[NOT NULL];*
*ime_tablice **IME_TIPA**;*

Razlikuje se od jednodimenzionalnog polja po tome što dinamički povećava broj redaka dok polje ima određenu granicu. Uz to kod brisanja ugnježđene tablice ona postaje šuplja, odnosno ako se obriše polje na indeksu 5, ono će ostati prazno kao što vidimo na slici 5.



Slika 5. Razlika između ugnježđene tablice i jednodimenzionalnog polja[33]

No, postoje još neke razlike s obzirom na asocijativni niz, a to je da kod drugog koraka deklariranja će doći do greške *ORA-06531: Reference to uninitialized collection* jer se sama kolekcija nije inicijalizirala, pa je dalje u kodu ona *null*. Kako bi se to izbjeglo koristi se konstruktor kako bi se kreirala prazna kolekcija koja nije *null*. U konstruktoru se mogu

[32] Rosenzweig B., Rakhimov E. (2009) – *Oracle PL/SQL by Example Fourth Edition*, str.319.
[33] Oracle (2009) - Oracle Database PL/SQL Language Reference 11g Release 1, *Understanding Nested Tables*, URL: http://docs.oracle.com/cd/B28359_01/appdev.111/b28370/collections.htm#i24239, dostupno 14.06.2014.

pridjeliti i neke vrijednosti, ali kod same deklaracije je rijetka situacija da se one znaju, pa se koristi prazni konstruktor kako je prikazano sljedećom sintaksom:

ime_varijable tip_podataka := tip_podataka()

I uz to prije svakog zapisivanja u kolekciju potrebno je proširiti njezinu veličinu, naredbom *ime_kolekcije.EXTEND* kako je pokazano u sljedećem primjeru.

```
DECLARE
        CURSOR c_servis IS
                SELECT * FROM servis;
        TYPE servisi IS TABLE OF servis.adresa_servisa%TYPE;
        tablica servisi := servisi();
BEGIN
        FOR r IN c_servis LOOP
                tablica.EXTEND;
                tablica(r.id_servisa) := r.adresa_servisa;
                DBMS_OUTPUT.PUT_LINE('Vrijednost: ' || tablica(r.id_servisa));
        END LOOP;
END;

Vrijednost: Ilica 58
Vrijednost: Radnička cesta 12
Vrijednost: Ulica braće Radić 2
```

9.3. Metode kolekcija[34]

Za rad s kolekcijama postoje metode koje se mogu koristiti, te je u prethodnom poglavlju korištena jedna od njih – EXTEND. Metode se koriste koristeći sintaksu

ime_kolekcija.ime_metode

Metode za rad s kolekcijama su sljedeće:

- EXISTS – metoda vraća TRUE ako postoji element pod tim indeksom
- COUNT – vraća ukupan broj elemenata kolekcije
- EXTEND – povećava veličinu kolekcije za jedno polje
- DELETE – metoda koja briše sve elemente kolekcije, elemente u nekom rasponu ili određeni element. Za napomenuti je da DELETE metoda ne obriše polja, nego samo vrijednosti tih polja.
- FIRST – metoda koja vraća prvi indeks kolekciji
- LAST – metoda koja vraća zadnji indeks u kolekciji
- PRIOR – metoda vraća element koji se nalazi na indeksu koji se nalazi prije proslijeđenog indeksa

[34] Rosenzweig B., Rakhimov E. (2009) – *Oracle PL/SQL by Example Fourth Edition*, str.322.

- NEXT – metoda vraća element koji se nalazi na indeksu koji slijedi nakon proslijeđenog indeksa
- TRIM – metoda koja briše jedan ili više elemenata s kraja liste, te ova metoda obriše vrijednosti i polja

```
DECLARE
        TYPE tip_kol IS TABLE OF NUMBER;
        kolekcija tip_kol := tip_kol(2,3,4);
BEGIN
        DBMS_OUTPUT.PUT_LINE('Zadnji indeks: ' || kolekcija.last());
        FOR i IN 1 .. kolekcija.LAST() LOOP
                DBMS_OUTPUT.PUT_LINE('Indeks ' || i ||': ' || kolekcija(i));
        END LOOP;
        kolekcija.TRIM(1);
        DBMS_OUTPUT.PUT_LINE('Zadnji indeks: ' || kolekcija.last());
        FOR i IN 1 .. kolekcija.LAST() LOOP
                DBMS_OUTPUT.PUT_LINE('Indeks ' || i ||': ' || kolekcija(i));
        END LOOP;
END;

Zadnji indeks: 3
Indeks 1: 2
Indeks 2: 3
Indeks 3: 4
Zadnji indeks: 2
Indeks 1: 2
Indeks 2: 3
```

Važno je za napomenuti da se metode TRIM i EXTEND ne mogu koristiti za asocijativne nizove. U primjeru iznad metoda TRIM obriše zadnji element kolekcije. Kad se koristi DELETE metoda može se koristiti metoda bez parametara koja obriše sve elemente kolekcije, DELETE(x) gdje je x indeks elementa koji se briše ili DELETE(x,y) gdje je su x i y raspon indeksa koji se brišu iz kolekcije.

9.4. Polja varijabilne duljine[35]

Polja varijabilne duljine su kolekcije koja su slična ugnježđenim tablicama, odnosno to je jednodimenzionalno polje koje ima ograničen maksimalni broj elemenata kako je i prikazano na slici 6.

[35] Rosenzweig B., Rakhimov E. (2009) – *Oracle PL/SQL by Example Fourth Edition*, str.334.

Slika 6. Polje varijabilne duljine[36]

Polje varijabilne duljine ima početni indeks koji počinje od 1 te se povećava za 1 sve do svoje maksimalne veličine. To ograničenje se definira kod deklaracije kolekcije sljedećom sintaksom:

```
TYPE ime_tipa IS {VARRAY | VARYING ARRAY} (maksimalna_veličina) OF
tip_elemenata [NOT NULL];
ime_polja IME_TIPA;
```

Princip deklariranja je sličan kao i kod ugnježđenih tablica te se odvija u dva koraka gdje prvo definiramo tip polja,a u drugom ga inicijaliziramo. Te ponovno moramo koristiti konstruktor kako kolekcija ne bi bila jednaka *null* nego prazna ili napunjena nekim inicijalnim vrijednostima.

```
DECLARE
        CURSOR c_servis IS
                SELECT * FROM servis;
        TYPE servisi IS VARRAY(10) OF servis.adresa_servisa%TYPE;
        polje servisi := servisi();
BEGIN
        FOR r IN c_servis LOOP
                polje.EXTEND;
                polje(r.id_servisa) := r.adresa_servisa;
        END LOOP;
        polje.TRIM(1);
        FOR i in 1 .. polje.LAST() LOOP
                DBMS_OUTPUT.PUT_LINE('Vrijednost: ' || polje(i));
        END LOOP;
END;

Vrijednost: Ilica 58
Vrijednost: Radnička cesta 12
```

U gornjem primjeru smo koristili TRIM metodu za brisanje iz polja varijabilne duljine, iz razloga što DELETE metoda ne radi jer je taj tip kolekcije je uzastopan te ne može imati rupe koje se uzrokuju tom metodom. Može se koristiti samo DELETE() za micanje svih elemenata kolekcije i metoda TRIM koja miče elemente s kraja kolekcije te tako ne uzrokuje rupe. Uz to ovdje postoji i metoda LIMIT() koja vraća ograničenje tog tipa kolekcije.

[36] Rosenzweig B., Rakhimov E. (2009) – *Oracle PL/SQL by Example Fourth Edition*, str.334.

9.5. Višeslojne kolekcije[37]

Oracle omogućava kreiranje kolekcija koje za tip podataka imaju drugu kolekciju. Primjerice se tako mogu definirati dvodimenzionalna polja, te je za pristupanje elementu tog polja potreban indeks prvog i drugog polja isto kao i u ostalim programskim jezicima.

```
DECLARE
        TYPE tip_1 IS VARRAY(2) OF INTEGER;
        TYPE tip_2 IS VARRAY(2) OF tip_1;
        v1 tip_1 := tip_1();
        v2 tip_2 := tip_2();
BEGIN
        v1 := tip_1(1,2);
        v2.EXTEND;
        v2(1) := v1;
        v1 := tip_1(3,4);
        v2.EXTEND;
        v2(2) := v1;
        FOR i IN 1 .. v2.LAST() LOOP
                DBMS_OUTPUT.PUT_LINE('Vanjski element - ' || i);
                FOR j IN 1 .. v2(i).LAST() LOOP
                        DBMS_OUTPUT.PUT_LINE('Element ('||i||','||j||') - '||v2(i)(j));
                END LOOP;
        END LOOP;
END;

Vanjski element - 1
Element (1,1) - 1
Element (1,2) - 2
Vanjski element - 2
Element (2,1) - 3
Element (2,2) - 4
```

U primjeru iznad je definirano v1 polje koje prima elemente tipa INTEGER, dok drugo polje v2 prima elemente koji su tipa prvog definiranog polja v1. Zatim se inicijalizira prvo polje v1 s vrijednostima (1,2) te se to polje doda u drugo polje v2, te se isti postupak ponovi još jednom za vrijednosti (3,4). Nadalje se ispis svih elemenata odvija kroz dvije FOR petlje gdje vanjska petlja prolazi kroz polje v2, a unutarnja petlja prolazi kroz elemente tog polja, odnosno polja v1. Tako je drugi element polja v2, polje v1 s vrijednostima (3,4) gdje se vrijednost 3 nalazi na lokaciji (2,1). U lokaciji (2,1), indeks 2 predstavlja drugi element polja v2, a indeks 1 predstavlja prvi element u polju v1.

[37] Oracle (2009) - Oracle Database PL/SQL Language Reference 11g Release 1, *Using Multidimensional Collections*, URL: http://docs.oracle.com/cd/B28359_01/appdev.111/b28370/collections.htm#LNPLS00507, dostupno 14.06.2014.

10. Zapis[38]

Zapis (*eng. Record*) je kompozitna struktura podataka koja grupira logički povezane podatke pohranjene u polja, gdje svako polje ima svoje ime i tip podatak. Najbliže je *struct* tipu podataka u jezicima C i C++.[39] PL/SQL omogućava deklariranje zapisa na tri načina:

- zapis temeljen na tablici (*eng. table-based record*)
- zapis temeljen na kursoru *(eng. cursor-based record)*
- korisnički definiran zapis *(eng. programmer-defined records*)

Zapis temeljen na tablici se deklarira pomoću %ROWTYPE atributa nad tablicom. Polja će imati ista imena i tip podataka kao i stupci u odabranoj tablici.

```
DECLARE
        table_based proizvodac%ROWTYPE;
BEGIN
        SELECT * INTO table_based
        FROM proizvodac where id_proizvodac = 4;
        DBMS_OUTPUT.PUT_LINE('Proizvođač: ' || table_based.ime_proizvodac);
END;
```

Proizvođač: Toshiba

Zapis temeljen na kursoru se također deklarira pomoću %ROWTYPE atributa, ali nad kursorom. Takva struktura podatak ima malo više slobode jer nije ograničena na polja samo jedne tablice već na polja svih tablica koje deklariraju u SELECT upitu kursora.

```
DECLARE
        CURSOR kursor IS
                SELECT p.ime_proizvodac, count(*) AS broj FROM nalog n
                JOIN proizvodac p ON p.id_proizvodac = n.proizvodac_id_proizvodac
                WHERE p.id_proizvodac < 4 GROUP BY p.ime_proizvodac;
        cursor_based kursor%ROWTYPE;
BEGIN
        FOR r IN kursor LOOP
                cursor_based := r;
                DBMS_OUTPUT.PUT_LINE('Za proizvođača: ' ||
                cursor_based.ime_proizvodac || ' je zaprimljeno ' ||  cursor_based.broj || '
                uređaja');
        END LOOP;
END;
```

Za proizvođača: Acer je zaprimljeno 263 uređaja

[38] ZenTut (2014) - PL/SQL, *PL/SQL Record*, URL: http://www.zentut.com/plsql-tutorial/plsql-record, dostupno 14.06.2014.
[39] Oracle (2005) - Oracle Database PL/SQL Language Reference 10g Release 2, *Record Definition*, URL: http://docs.oracle.com/cd/B19306_01/appdev.102/b14261/record_definition.htm, dostupno 14.06.2014.

Za proizvođača: Asus je zaprimljeno 402 uređaja
Za proizvođača: Hewlett-Packard je zaprimljeno 387 uređaja

Korisnički definiran zapis se deklarira u dva koraka. Prvo je potrebno definirati tip koristeći TYPE te unutar njega definirati sva polja koja će se koristiti, pomoću sljedeće sintakse:

```
TYPE ime_zapisa IS RECORD (deklaracija_polja[,deklaracija_polja]...);
```

Kod deklaracije samog polja mogu se koristiti varijable skalarnog tipa podataka, usidrenih tipova podataka, podtipovi, kolekcije, kursori čak i drugi zapisi. Ovakav zapis nije ograničen samo na polja koja se nalaze u tablici ili kursoru te tako omogućava kreiranje dodatnih varijabli te njihovo povezivanje u logičku cjelinu. U drugom koraku se definira varijabla tog istog tipa te se u nju zapisuju vrijednosti.

```
DECLARE
        TYPE tip IS RECORD(
                ime proizvodac.ime_proizvodac%TYPE,
                broj NUMBER);
        zapis tip;
BEGIN
        SELECT p.ime_proizvodac, count(*)
        INTO zapis FROM nalog n
        JOIN proizvodac p ON p.id_proizvodac = n.proizvodac_id_proizvodac
        WHERE p.id_proizvodac = 4 GROUP BY p.ime_proizvodac;

        DBMS_OUTPUT.PUT_LINE('Za proizvođača: ' || zapis.ime || ' je zaprimljeno ' ||
        zapis.broj || ' uređaja');
END;
```

Za proizvođača: Toshiba je zaprimljeno 367 uređaja

U gornjem primjeru je definiram tip podataka *tip* sa dva polja te varijabla tog tipa podataka. U izvršnom dijelu bloka je pomoću SELECT upita u varijablu zapisa upisano ime proizvođača i broj uređaja koji su zaprimljeni na servis. Za zapis se moraju spomenuti još sljedeće mogućnosti:

- varijabla zapisa se može upisati u drugu varijablu zapisa, pod uvjetom da imaju isti broj polja i da su ona istog tipa podataka.
- Zapis može biti korišten kao parametar u funckiji
- Zapis može biti korišten kao izlazna vrijednost funkcije
- Za provjeru da li je zapis *null* se moraju provjeriti sva njegova polja da li su *null*
- Za usporedbu dva zapisa moraju se usporediti sva njegova polja pojedinačno

11. Iznimke

U PL/SQL-u se mogu javiti dvije vrste iznimaka (*eng. Exceptions*) i to su iznimke koje se javljaju kod kompajliranja samog koda i iznimke koje se javljaju kod izvođenja programa. Iznimke kod kompajliranja se javljaju kad se pojavljuje pogreška u sintaksi. U ovom poglavlju će se obraditi iznimke koje se javljaju kod izvođenja programa. Za takve iznimke postoji poseban dio unutar PL/SQL bloka u kojem se one obrađuju.[40] Obrada podrazumijeva naredbe koje se izvode u slučaju da dođe do iznimke, te dio za obradu iznimaka izgleda kao:

```
EXCEPTION
        WHEN ime_iznimke THEN
                naredbe_za_izvršiti_u_slučaju_greške
```

No za primjer prihvata i obrade jedne iznimke ispod je kod u kojem se upitom ne vraća ni jedan redak za upis u varijablu. Zbog toga dolazi do iznimke NO_DATA_FOUND.

```
DECLARE
        naziv VARCHAR2(20);
BEGIN
        SELECT naziv_kvara INTO naziv
        FROM kvar WHERE id_kvara = 30;
        DBMS_OUTPUT.PUT_LINE('Kvar: ' || naziv);
EXCEPTION
        WHEN NO_DATA_FOUND THEN
                DBMS_OUTPUT.PUT_LINE('Nema te kategorije kvara');
END;

Nema te kategorije kvara
```

No zato jer je kod pripremljen na mogućnost da dođe do te iznimke, on ga prihvaća i ispisuje odgovarajuću poruku. Ovdje bi se primjerice mogao koristiti neki *log* u koji bi se upisivale iznimke ili neka druga odgovarajuća naredba. Da iznimka nije prihvaćena, nakon nje bi se kod prestao izvršavati. Važno je za napomenuti da se nakon dijela za obradu iznimaka, kontrola se ne vraća izvršnom dijelu nego blok završava. U PL/SQL-u postoje dvije vrste iznimaka koje se mogu prihvatiti i obraditi:

- ugrađene iznimke (*eng. Built-in Exceptions*)
- iznimke definirane od strane korisnika (*eng. User-defined Exceptions*)

[40] Rosenzweig B., Rakhimov E. (2009) – *Oracle PL/SQL by Example Fourth Edition*, str.165.

11.1. Ugrađene iznimke

Oracle PL/SQL ima svoja pravila za koja se implicitno provjerava ispravnost tokom izvođenja koda. Ako program prekrši neko od tih pravila, implicitno se okida iznimka. Kod ugrađenih iznimaka zbog njihovog implicitnog okidanja, je potrebno samo definirati akcije koje se izvršavaju u tom slučaju. U prethodnom poglavlju je korištena iznimka NO_DATA_FOUND koja se okida ako se ne vrati ni jedan redak. Takvih pravila ima više te svako od njih ima naziv koji se koristi na isti način. Neke od PL/SQL ugrađenih grešaka su[41]:

- ACCESS_INTO_NULL – kada se *null* objektu dodjeljuje vrijednost
- CASE_NOT_FOUND – kada ni jedan WHEN u CASE selekciji nije odabran, a ne postoji ELSE klauzula
- COLLECTION_IS_NULL – kada se pokuša izvršiti metoda kolekcije nad *null* kolekcijom
- CURSOR_ALREADY_OPEN – kada se pokuša otvoriti već otvoreni kursor
- DUP_VAL_ON_INDEX – kod pokušavanja spremanja duplicirane vrijednosti u kolumnu koja ima UNIQUE ograničenje nad sobom
- INVALID_CURSOR – kod nedopuštenih operacija nad kursorom, primjerice zatvaranje neotvorenog kursora
- INVALID_NUMBER – kod konverzije teksta u broj, ako tekst ne predstavlja ispravan broj
- LOGIN_DENIED – kod pokušavanja spajanja na bazu s neispravnim imenom i lozinkom
- NO_DATA_FOUND – kada SELECT naredba ne vrati ni jedan redak
- NOT_LOGGED_ON – kod poziva na bazu bez uspostavljene veze s bazom
- PROGRAM_ERROR – kod PL/SQL internih problema
- ROWTYPE_MISMATCH – kod FETCH-a iz kursora u varijablu nekomplatibilnog tipa podataka
- STORAGE_ERROR – kod nedostatka ili greške u memoriji
- TOO_MANY_ROWS – kada SELECT INTO naredba vrati više od jednog retka
- VALUE_ERROR – kod nedopuštene operacije s varijablama, nedopuštene konverzije ili zapisivanja vrijednosti pogrešne veličine
- ZERO_DIVIDE – kod pokušaja djeljenja s nulom

[41] Oracle (2014) - Oracle Database PL/SQL Language Reference 11g Release 2, *Predefined Exceptions*, URL: http://docs.oracle.com/cd/E11882_01/appdev.112/e25519/errors.htm#LNPLS00703, dostupno 14.06.2014.

11.2. Iznimke definirane od strane korisnika[42]

U svakom programu se mogu javiti specifične greške koje nisu pokrivene od strane PL/SQL-a pa je tako potrebno definirati izminke koje se mogu pozivati u tim slučajevima. Isto kao i kod ugrađenih iznimaka, u dijelu za obradu iznimaka se definiraju akcije koje se izvršavaju u tom slučaju. Da se takva iznimka može koristiti prvo se treba deklarirati u deklarativnom dijelu PL/SQL bloka sljedećom sintaksom:

```
DECLARE
        ime_iznimke EXCEPTION;
BEGIN
        ...
        IF uvjet THEN
                RAISE ime_iznimke;
        END IF;
        ...
EXCEPTION
        WHEN ime_iznimke THEN
                naredbe_za_obradu_iznimaka
END;
```

Deklaracija iznimaka je veoma slična deklaraciji varijabli, samo što se nakon imena navodi tip podataka nego ključna riječ EXCEPTION koja označuje da se radi o iznimci. Uobičajena praksa je za ime iznimke koristiti prefiks *e_* kako bi se mogla razlikovati od ostalih varijabli. Sljedeći korak rada sa iznimkama definiranim od strane korisnika je okidanje same iznimke. Za razliku od ugrađenih iznimaka, ove je potrebno provjeriti da li je ispunjen uvjet za okidanje iznimke. Ako je uvjet ispunjen, odnosno TRUE okinut će se iznimka ključnom riječi RAISE te se ista prihvatiti u dijelu za obradu iznimaka.

```
DECLARE
        broj_naloga NUMBER;
        nema_prodaje EXCEPTION;
BEGIN
        SELECT COUNT(*) INTO broj_naloga
        FROM nalog WHERE zaposlenik_id_zap = 10;
        IF broj_naloga = 0 THEN
                RAISE nema_prodaje;
        ELSE
                UPDATE zaposlenik SET placa_zap = placa_zap * 1.1
                WHERE id_zap = 10;
                DBMS_OUTPUT.PUT_LINE('Plaća povećana');
        END IF;
EXCEPTION
        WHEN nema_prodaje THEN
                UPDATE zaposlenik SET placa_zap = placa_zap * 0.9
                WHERE id_zap = 10;
```

[42] Rosenzweig B., Rakhimov E. (2009) – *Oracle PL/SQL by Example Fourth Edition*, str.188.

```
            DBMS_OUTPUT.PUT_LINE('Plaća smanjena');
        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('Došlo je do greške');
END;
```

Plaća smanjena

Za potrebe gornjeg primjera u bazu je dodan novi zaposlenik koji nema ni jedan obrađeni nalog. Iz baze se za tog zaposlenika vadi broj obrađenih naloga te ako nema ni jedan nalog, dolazi do smanjenja plaće. Odnosno ako je broj naloga 0, dolazi do okidanja iznimke koja pokreće naredbu koja smanjuje plaću tom zaposleniku.

Važno je ovdje napomenuti još nekoliko stvari, a to je da SELECT upiti s grupnim funkcijama kao COUNT, MAX, SUM iako nema ni jednog retka ne okidaju ugrađenu iznimku NO_DATA_FOUND jer oni vraćaju 0. Ovdje je među iznimkama korištena iznimka OTHERS koja prima sve iznimke koje nisu specifično definirane da budu prihvaćene.[43] Što je dobra praksa za uključiti u svaki PL/SQL blok, kako bi se osigurali od neočekivanih grešaka.

```
DECLARE
    radnik zaposlenik%ROWTYPE;
BEGIN
    SELECT * INTO radnik
    FROM zaposlenik WHERE id_zap = 11;
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Greška '||SQLCODE||' - '||SQLERRM);
END;
```

Greška 100 - ORA-01403: no data found

U primjeru iznad je kod dohvata zaposlenika dohvaćeno 0 redaka odnosno okinuta je iznimka NO_DATA_FOUND. U dijelu za obradu grešaka se ona prihvatila kroz iznimku OTHERS, ali tada kako bi znali koji je kod i poruka greške se koriste funkcije SQLCODE za dohvat koda greške i SQLERRM za dohvat poruke greške. Što olakšava analiziranje grešaka koje se javljaju i zapisivanje istih u neki *log*.

11.3. Propagacija iznimaka[44]

Vidljivost iznimaka u ugnježđenim blokovima je ista kao vidljivost varijabli. Nakon vidljivosti iznimaka treba se urediti i njihova propagacija, odnosno njihovo kretanje sve dok se one ne prihvate. Kada dođe do iznimke u unutarnjem bloku, iznimka može:

[43] Tech On The Net (2014) - Oracle/ PLSQL, *WHEN OTHERS Clause*, URL: http://www.techonthenet.com/oracle/exceptions/when_others.php, dostupno 14.06.2014.
[44] Oracle (2014) - Oracle Database PL/SQL Language Reference 11g Release 2, *Exception Propagation*, URL: http://docs.oracle.com/cd/E11882_01/appdev.112/e25519/errors.htm#LNPLS00706, dostupno 14.06.2014.

- ne propagirati
- propagirati iz unutarnjeg u vanjski blok
- neobrađena iznimka se predaje okolini

Iznimka ne propagira u slučaju da se okine u unutarnjem bloku, te se u dijelu za prihvat iznimaka istog bloka nalazi definiran prihvat. Tada se kaže da iznimka nije propagirala jer se njezin prihvat i obrada obavila u istom bloku gdje je i nastala. Iznimka propagira kada do iznimke dođe u unutarnjem bloku, te u tom bloku nema dijela za prihvat te greške. Tada se ona proslijeđuje vanjskim blokovima sve dok ne dođe do dijela za prihvat.

```
BEGIN
        DECLARE
                broj NUMBER;
        BEGIN
                SELECT ime_zap INTO broj FROM zaposlenik WHERE id_zap = 2;
        EXCEPTION
                WHEN NO_DATA_FOUND THEN
                        DBMS_OUTPUT.PUT_LINE('Nema zaposlenika');
        END;
EXCEPTION
        WHEN VALUE_ERROR THEN
                DBMS_OUTPUT.PUT_LINE('Kriva vrijednost');
        WHEN OTHERS THEN
                DBMS_OUTPUT.PUT_LINE('Došlo je do greške');
END;
```

U gornjem primjeru se vidi da je u unutarnjem bloku došlo do greške, tj. VALUE_ERROR jer se tekst pokušao zapisati u brojčanu varijablu. U unutarnjem bloku nije bilo dijela za prihvat te greške, te se ona propagirala prema van. U vanjskom bloku je postojala obrada te iznimke te se ona prihvatila u tom dijelu. Općenito vrijedi za greške u ugnježđenim blokovima da se greška šalje prema van, sve dok se ona ne prihvati ili ako se ne prihvati onda se predaje kao neobrađena okolini gdje je pokrenut blok. Prekidanje programa zbog neobrađenih grešaka je nepoželjna situacija, pa je dobra praksa koristiti OTHERS barem u vanjskom bloku kako bi se sve eventualne greške prihvatile i na neki način obradile i završile blok na ispravan način.

11.4. Ponovno dizanje iznimaka[45]

U ugnježđenim blokovima moguće je i ponovno okinuti iznimku. Primjerice ako je u vanjskom bloku deklarirana iznimka, te se ona pozove naredbom RAISE u unutarnjem bloku. U unutarnjem bloku postoji prihvat te greške te se unutar tog prihvata iznimka može ponovno

[45] Rosenzweig B., Rakhimov E. (2009) – *Oracle PL/SQL by Example Fourth Edition*, str.201.

okinuti koristeći samo naredbu RAISE bez navođenja imena iznimke. Ta iznimka će se prihvatiti u vanjskom bloku.

```
DECLARE
        broj NUMBER;
        iznimka EXCEPTION;
BEGIN
        BEGIN
                SELECT COUNT(*) INTO broj FROM nalog WHERE zaposlenik_id_zap =
                11;
                IF broj > 0 THEN
                        DBMS_OUTPUT.PUT_LINE('Prodaja: ' || broj);
                ELSE
                        RAISE iznimka;
                END IF;
        EXCEPTION
                WHEN iznimka THEN
                        RAISE;
        END;
EXCEPTION
        WHEN iznimka THEN
                DBMS_OUTPUT.PUT_LINE('Nema prodaje');
END;

Nema prodaje
```

11.5. PRAGMA EXCEPTION_INIT[46]

Postoje Oracle definirane greške koje nemaju svoje ime nego samo svoj kod, te se takve greške ne mogu prihvatiti pomoću njihovog imena. Tu dolazi naredba *PRAGMA EXCEPTIO_INIT(ime_iznimke, kod_iznimke)*.

```
DECLARE
        zastoj EXCEPTION;
        PRAGMA EXCEPTION_INIT(zastoj, -60);
BEGIN
        naredbe koje dovode do iznimke zastoj (deadlock)
EXCEPTION
        WHEN zastoj THEN
                naredbe za obradu zastoja
END;
```

U primjeru[47] iznad je uzeta greška s kodom -60 koja predstavlja zastoj (*eng. deadlock*) te da se ona može prihvatiti prvo se mora definirati iznimka pod nekim imenom. U drugom koraku se

[46] Rosenzweig B., Rakhimov E. (2009) – *Oracle PL/SQL by Example Fourth Edition*, str.217.
[47] Oracle (2005) - Oracle Database PL/SQL Language Reference 10g Release 2, *Associating a PL/SQL Exception with a Number*, URL: http://docs.oracle.com/cd/B19306_01/appdev.102/b14261/errors.htm#i3329, dostupno 14.06.2014.

u PRAGMA EXCEPTION_INIT upiše ime iznimke koja je prethodno deklarirana i kod -60 te se tek tada ta greška može prihvatiti i obraditi pod imenom deklarirane iznimke.

12. Potprogrami

Do sada su u primjerima korišteni samo anonimni blokovi koji su bili pokretani kao skripte i bili kompajlirani kod pokretanja. Sada će se koristiti imenovani blokovi, odnosno potprogrami. PL/SQL potprogram je imenovani PL/SQL blok koji se može ponovno pozivati. Potprogrami mogu imati ulazne i izlazne parametre, koji se mogu razlikovati kod svakog pozivanja.[48] Neke od prednost kod korištenja potprograma su[49]:

- modularnost – svaki potprogram je mali modul koji se može lakše održavati i koristiti
- lakše oblikovanje aplikacije – lakše je testirati i mjenjati aplikaciju po malim modulima
- lakše održavanje – promjena samo jednog potprograma bez promjene koda gdje se on poziva
- korištenje paketa – pisanje povezanih potprograma unutar jednog paketa koji služi kao sučelje prema potprogramima čija je namjena usko povezana
- ponovno korištenje (*eng. reusability*) – pozivanje istih potprogram više puta
- bolje performanse – jer se potprogrami prevode u strojni kŏd kod kreiranja i promjene, pa ih nije potrebno kompajlirati kod svakog poziva

Potprogrami se mogu kreirati unutar PL/SQL bloka – ugnježđeni potprogram (*eng. nested subprogram*), unutar paketa (*eng. package subprogram*), te na razini sheme (*eng. standalone subprogram*). PL/SQL omogućava dvije vrste potprograma:

- funkcije – potprogrami koji imaju povratnu vrijednost, koja se vraća nakon izvršenja potprograma
- procedure – potprogrami koji nemaju povratnu vrijednost, te se koriste samo za izvršavanje određenih akcija

Sam potprogram se sastoji od zaglavlja u kojem se specificira ime potprograma i lista parametara koje prima potprogram. Lista parametara je opcionalan dio. Nakon toga dolaze tri dijela kao i u anonimnom bloku, odnosno deklarativan, izvršan i dio za obradu iznimaka.

[48] Oracle (2014) - Oracle Database PL/SQL Language Reference 11g Release 2, *PL/SQL Subprograms*, URL: http://docs.oracle.com/cd/E11882_01/appdev.112/e25519/subprograms.htm#LNPLS008, dostupno 14.06.2014.

[49] Oracle (2014) - Oracle Database PL/SQL Language Reference 11g Release 2, *Reasons to Use Subprograms*, URL: http://docs.oracle.com/cd/E11882_01/appdev.112/e25519/subprograms.htm#LNPLS99900, dostupno 14.06.2014.

Jedina razlika je ta da deklarativan dio ne započinje s ključnom riječi DECLARE nego se ona izostavlja pa se varijable deklariraju između zaglavlja i ključne riječi BEGIN.

12.1. Procedure[50]

Procedura je potprogram koji može primati parametre, ali ne vraća vrijednosti. Kreira se koristeći sljedeću sintaksu:

```
CREATE [OR REPLACE] PROCEDURE ime_procedure
[(ime_parametra [IN | OUT | IN OUT] tip [, ...])]
{IS | AS}
BEGIN
        < tijelo_procedure>
END ime_procedure;
```

U toj sintaksi *ime_procedure* predstavlja ime pod kojim će se ona kreirati u bazi, te pomoću kojega će se procedura pozivati. Moguće je koristiti i opcionalni dio *OR REPLACE* koji služi da ako procedura već postoji da se postojeća procedura prebriše s novom, izmjenjenom procedurom. Nakon toga se definira lista parametara ako trebamo parametre, a poslije toga dolazi ključna riječ *AS* ili *IS*. Svejedno je koju od njih koristimo jer su to sinonimi. Kada je definirano zaglavlje procedure, nadalje ide standardna struktura bloka osim što se za deklarativni dio ne piše riječ *DECLARE* nego on započinje nakon ključnih riječi *AS* ili *IS*.

```
CREATE OR REPLACE PROCEDURE povecanjePlace
(minimalnoNaloga IN NUMBER, povecanje IN BINARY_FLOAT) IS
        CURSOR c_zaposlenici IS
                SELECT * FROM zaposlenik
                FOR UPDATE;
        broj NUMBER;
BEGIN
        FOR r IN c_zaposlenici LOOP
                SELECT COUNT(*) INTO broj FROM nalog WHERE zaposlenik_id_zap =
                r.id_zap;
                IF broj > minimalnoNaloga THEN
                        UPDATE zaposlenik SET placa_zap = placa_zap * povecanje
                        WHERE CURRENT OF c_zaposlenici;
                        DBMS_OUTPUT.PUT_LINE(r.ime_zap||' '||r.prezime_zap||' je dobio
                        povećanje plaće za koeficijent '||TO_CHAR(povecanje, '9.99'));
                ELSE
                        DBMS_OUTPUT.PUT_LINE(r.ime_zap||' '||r.prezime_zap||' nije
                        dobio povišicu');
                END IF;
        END LOOP;
END povecanjePlace;
```

[50] Rosenzweig B., Rakhimov E. (2009) – *Oracle PL/SQL by Example Fourth Edition*, str.441.

```
--pozivanje procedure u drugom bloku
BEGIN
        povecanjePlace(300,1.15);
END;

Ivica Ivic nije dobio povišicu
Marija Maric je dobio povećanje plaće za koeficijent  1.15
Marko Markic je dobio povećanje plaće za koeficijent  1.15
Kruno Krunic je dobio povećanje plaće za koeficijent  1.15
Matija Matic je dobio povećanje plaće za koeficijent  1.15
Kreso Kresic je dobio povećanje plaće za koeficijent  1.15
Petar Petric je dobio povećanje plaće za koeficijent  1.15
Slavko Slavic je dobio povećanje plaće za koeficijent  1.15
Sanja Sanjic nije dobio povišicu
Marinko Maric nije dobio povišicu
```

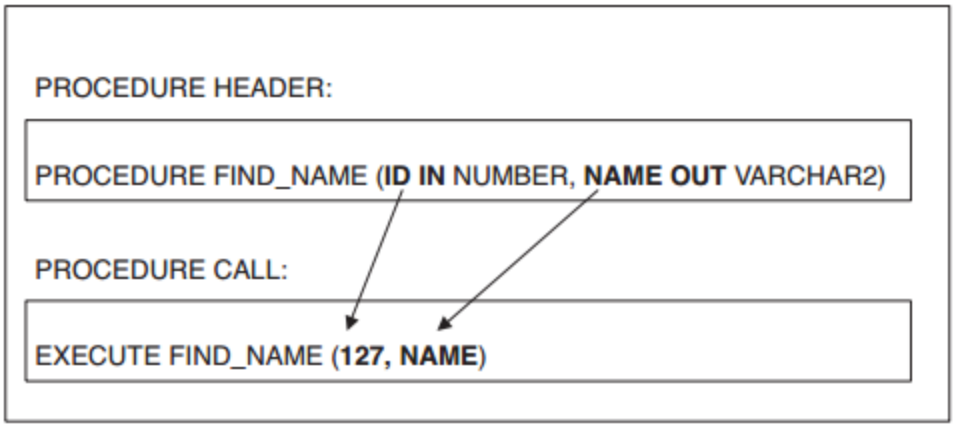
U primjeru iznad je definirana procedura koja svim zaposlenicima daje povišicu, ovisno o tome koliko su naloga obradili. Tako procedura prima dva parametra i to su *minimalnoNaloga* i *povecanje*. Prvi parametar označuje koliko naloga mora imati zaposlenik da mu se poveća plaća dok drugi označuje koeficijent za koji se povećava plaća. U kursor se dohvate svi zaposlenici te se kroz FOR petlju prolazi kroz sve i za svakoga dobavlja broj naloga. Kad se dohvati broj naloga provjeri se da li je veći od potrebnog broja i ako je uvjet istinit, povećava se plaća. Takva procedura je pohranjena, te se ona poziva u drugom PL/SQL bloku samo navodeći njezino ime i proslijeđujući joj parametre *povecanjePlace(300,1.15).* U ovom slučaju se izvodi procedura s parametrima gdje je 300 minimalni broj naloga koje mora imati zaposlenik da bi mu se povećala plaća za koeficijent 1.15, odnosno 15 posto.

Ako je potrebno procedura se jednostavno može obrisati koristeći naredbu *DROP PROCEDURE ime_procedure*.

12.2. Vrste parametara u potprogramima[51]

Pomoću parametara se vrijednosti proslijeđuju između potprograma i programa iz kojeg se on poziva. To su vrijednosti koje se obrađuju i vraćaju kroz izvođenje procedure. Formalni parametri su ona imena koja se specificiraju u zaglavlju potprograma, dok su pravi parametri vrijednosti ili izrazi koji se pišu kod pozivanja potprograma. Formalni i pravi parametri moraju biti istog tipa podataka kao što se vidi na slici 7.

[51] Rosenzweig B., Rakhimov E. (2009) – *Oracle PL/SQL by Example Fourth Edition*, str.444.



Slika 7. Zaglavlje procedure i pozivanje iste[52]

Postoje tri vrste parametara koje se šalju potprogramu, a to su *IN*, *OUT* i *IN OUT*. *IN* vrsta parametra potprogramu prosljeđuje vrijednost koja se može se samo čitati (*eng. read-only*).[53] Unutar potprograma se ponaša kao konstanta, odnosno ne može mu se dodjeliti druga vrijednost. Takva vrsta parametara može biti bilo koja konstanta, varijabla ili literal. Proslijeđuje se referenca na parametar.

OUT vrsta parametra potprogramu prosljeđuje vrijednost putem varijable, te se ona nakon obrade referencira na varijablu u programu koji je pozvao potprogram. Parametar vrste OUT se unutar potprograma ponaša kao varijabla te se može mijenjati. Mora se poslati samo kao varijabla. Ako bi primjerice pokušali kod poziva potprograma za OUT parametar proslijediti literalnu vrijednost dobili bi grešku *ORA-06550: expression '5' cannot be used as an assignment target*.

```
DECLARE
    x NUMBER := 4;
    y NUMBER := 0;
    PROCEDURE kvadrat(a IN NUMBER, b OUT NUMBER) IS
            BEGIN
                    b := a * a;
            END;
BEGIN
    kvadrat(x,y);
    DBMS_OUTPUT.PUT_LINE('Kvadrat od '||x||' je '||y);
    kvadrat(6,y);
    DBMS_OUTPUT.PUT_LINE('Kvadrat od 6 je '||y);
END;
```

[52] Rosenzweig B., Rakhimov E. (2009) – *Oracle PL/SQL by Example Fourth Edition*, str.444.
[53] Oracle (2014) - Oracle Database PL/SQL Language Reference 11g Release 2, *Subprogram Parameters*, URL: http://docs.oracle.com/cd/E11882_01/appdev.112/e25519/subprograms.htm#LNPLS00806, dostupno 14.06.2014.

```
Kvadrat od 4 je 16
Kvadrat od 6 je 36
```

U ovom primjeru, odnosno bloku se nalazi procedura za izračun kvadrata nekog broja. Procedura ima dva parametra, jedan tipa IN i drugi tipa OUT. Kod prvog poziva procedure proslijeđuje se varijabla kao IN parametar i varijabla kao OUT parametar. Nakon obrade procedura rezultat, odnosno kvadrat zapisuje u varijablu *y*. Kod drugog poziva je sve isto, samo što se kao *IN* parametar proslijeđuje literal.

IN OUT vrsta parametra potprogramu proslijeđuje vrijednost koja se nakon obrade vraća programu koji je pozvao potprogram. Također mora biti samo varijabla.

```
DECLARE
        x NUMBER := 4;
        PROCEDURE kvadrat(a IN OUT NUMBER) IS
                BEGIN
                        a := a * a;
                END;
BEGIN
        kvadrat(x);
        DBMS_OUTPUT.PUT_LINE('Kvadrat je '||x);
END;

Kvadrat je 16
```

U primjeru iznad je u proceduru kvadrat poslan samo jedan parametar kao *IN OUT* te se nakon obrade vrijednost zapisala u tu istu varijablu, odnosno u varijablu *x*.

12.3. Vrste proslijeđivanja[54]

Ako potprogram ima parametre koje prima, oni mu se moraju i proslijediti, te za to postoje tri različite vrste. Najpoznatije je *pozicijsko proslijeđivanje* (*eng. positional notation*), odnosno kako su parametri definirani u zaglavlju potprograma tako se i šalju kod njegovog proslijeđivanja. Recimo da imamo deklaraciju funkcije kao *ime_funkcije(x IN NUMBER; y IN NUMBER, z OUT NUMBER)* pozicijsko proslijeđivanje se radi kao *ime_funkcije(a, b, c)* gdje parametar *a* zamjenjuje formalni parametar *x*, *b* zamjenjuje *y* i *c* zamjenjuje *z*.

Postoji još *imenovno proslijeđivanje* (*eng. named notation*) gdje se pravi parametar zamjenjuje formalnim parametrom koristeći simbol =>, odnosno *formalni parametar => pravi parametar*. Pa bi tako poziv gore navedene funkcije glasio *ime_funkcije(x=>a, z=>c, y=>b)*. Važno je za napomenuti da kod ovakvog načina rada nije važna pozicija parametara.

[54] TutorialsPoint (2014) - PL/SQL Tutorial, *PL/SQL - Procedures*, URL: http://www.tutorialspoint.com/plsql/plsql_procedures.htm, dostupno 14.06.2014

Uz ove dvije vrste postoji još *mješovito proslijeđivanje* (*eng. mixed notation*) gdje se mogu koristit gore dvije navedene vrsta sa samo jednim ograničenjem, a to je da se uvijek prvo mora koristiti *pozicijsko proslijeđivanje* pa zatim *imenovno proslijeđivanje*.

12.4. Funkcije[55]

Funkcije su potprogrami koji primaju parametre te vraćaju jednu vrijednost. Kreiraju se sljedećom sintaksom:

```
CREATE [OR REPLACE] FUNCTION ime_funkcije
[(ime_parametra [IN | OUT | IN OUT] tip [, ...])]
RETURN tip_podataka_koji_se_vrati
{IS | AS}
BEGIN
        < tijelo_procedure >
END [ime_funkcije];
```

U sintaksi *ime_funkcije* je ime funkcije pod kojim se ona kreira u bazi, te kasnije poziva. Kao i kod procedure, opcionalnom naredbom OR REPLACE se može promjeniti postojeća funkcija. Nakon toga dolazi lista parametara i zatim ključna riječ RETURN i tip podataka koji će se vratiti. Svaka funkcija obavezno mora imati RETURN naredbu i tip podataka koji se vraća, te je to glavna razlika između procedure i funkcije. Nakon toga dolazi ključna riječ AS ili IS i nakon toga deklarativni dio funkcije sve do ključne riječi BEGIN nakon koje počinje tijelo funkcije.

```
CREATE OR REPLACE FUNCTION analitika
RETURN VARCHAR2 IS
        proizvodac VARCHAR2(30);
BEGIN
        SELECT ime INTO proizvodac FROM (
                SELECT p.ime_proizvodac AS ime, COUNT(*) AS broj FROM nalog n
                JOIN proizvodac p ON p.id_proizvodac = n.proizvodac_id_proizvodac
                GROUP BY p.ime_proizvodac ORDER BY broj DESC)
        WHERE ROWNUM = 1;

        RETURN proizvodac;
END;

BEGIN
        DBMS_OUTPUT.PUT_LINE('Zaprimljeno je najviše uređaja: '||analitika());
END;

Zaprimljeno je najviše uređaja: Asus
```

[55] Rosenzweig B., Rakhimov E. (2009) – *Oracle PL/SQL by Example Fourth Edition*, str.450.

U gornjem primjeru prvo je kreirana funkcija koja analizira sve zaprimljene naloge te vraća proizvođača, čijih je najviše uređaja zaprimljeno na servis. Definirano je da je povratna vrijednost funkcije VARCHAR2. U deklarativnom dijelu funkcije je zato kreirana varijabla tipa VARCHAR2(30) koja će se vratiti. Za naglasiti je da se oganičenje veličine ne upisuje kod definiranja povratnog tipa nego se ono implicitno postavlja prema tipu varijable koja se vraća. Sljedeće se upitom dohvaća naziv proizvođača. No prvo, Oracle ima malo nezgrapno definirano kako odrediti broj redaka koji će se dohvatiti. Iz tog razloga se koristi ugnježđeni upit gdje se u unutarnjem upitu dohvate svi proizvođači i broj naloga, te se taj rezultat sortira prema broju naloga od većeg do manjeg. Nakon toga se vanjskim upitom dohvaća samo prvi redak tog unutarnjeg upita pomoću ROWNUM. Na taj način se dobavlja ime proizvođača koji ima najviše uređaja na servisu te zapisuje u varijablu koja se zatim vraća pomoću RETURN. Nakon toga slijedi drugi korak, odnosno pozivanje funkcije u drugom bloku. U tom bloku se funkcija poziva direktno u naredbi za ispis samo navodeći njezino ime *analitika()*.

Jedna od dobrih praksi kod korištenja funkcija je da se koriste samo IN parametri, a ne OUT i IN OUT. Idealno je da funkcija prima nula ili više *IN* parametara i vraća jednu vrijednost RETURN naredbom.[56]

[56]Oracle (2014) - Oracle Database PL/SQL Language Reference 11g Release 2, *Subprogram Parameter Modes*, URL: http://docs.oracle.com/cd/E11882_01/appdev.112/e25519/subprograms.htm#LNPLS659, dostupno 14.06.2014.

13. Paketi

Paket je shema objekata koja grupira logički povezane PL/SQL tipove, varijable i potprograme. Sam paket se sastoji od dva obavezna dijela:

- specifikacija paketa (*eng. package specification*)
- tijelo ili definicija paketa (*eng. package body or definition*)

13.1. Specifikacija paketa[57]

Specifikacija paketa je sučelje (*eng. interface*) prema paketu, koje sadrži informacije o sadržaju paketa bez koda potprograma. U njemu se samo deklariraju tipovi, varijable, konstante, kursori i potprogrami. Sve to će se moći dohvatiti izvan paketa, odnosno svi objekti u specifikaciji paketa su javni objekti. Dok je svaki potprogram koji nije u specifikaciji paketa, a njegov kod se nalazi u tijelu paketa je privatni objekt te njemu mogu pristupati samo drugi potprogrami paketa. Specifikacija paketa se kreira sljedećom sintaksom:

```
CREATE PACKAGE ime_paketa
IS
        [deklaracija varijabli i tipova]
        [specifikacija kursora]
        [specifikacija potprograma]
END [ime_paketa];
```

13.2. Tijelo paketa[58]

Tijelo paketa sadrži izvršni kod za objekte koji se nalaze u specifikaciji paketa, kao i kod za potprograme koji se nalaze u specifikaciji i privatne potprograme, odnosno one koji nisu navedeni u specifikaciji. Sintaksa za tijelo paketa je sljedeća:

```
CREATE PACKAGE BODY ime_paketa
IS
        [deklaracija varijabli i tipova]
        [specifikacija i SELECT naredba kursora]
        [specifikacija i tijelo potprograma]
[BEGIN
        izvršne_naredbe]
[EXCEPTION
        obrada_iznimaka]
END [ime_paketa];
```

[57] Rosenzweig B., Rakhimov E. (2009) – *Oracle PL/SQL by Example Fourth Edition*, str.459.
[58] Rosenzweig B., Rakhimov E. (2009) – *Oracle PL/SQL by Example Fourth Edition*, str.460.

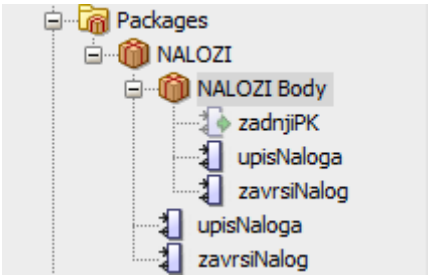
Kod tijela paketa treba pripaziti da kod kursora i potprograma se navede isto zaglavlje i definicija kao i u specifikaciji paketa. Svaka varijabla, tip, iznimka i konstanta deklarirana u specifikaciji može koristiti u tijelu, te ih nije potrebno ponovno deklarirati. Kada postoji paket, do njegovih elemenata se dolazi korištenjem sintakse *ime_paketa.element*.

```
CREATE OR REPLACE PACKAGE nalozi AS
        PROCEDURE upisNaloga
                (kvar NUMBER,
                proizvodac NUMBER,
                zaposlenik NUMBER,
                ime nalog.ime_podnositelja%TYPE,
                prezime nalog.prez_podnositelja%TYPE,
                telefon nalog.tel_podnositelja%TYPE,
                serijski nalog.serijski_proizvoda%TYPE,
                garancija nalog.garancija%TYPE);
        PROCEDURE zavrsiNalog
                (sifra NUMBER,
                cijena nalog.cijena%TYPE);
END nalozi;

CREATE OR REPLACE PACKAGE BODY nalozi AS
        FUNCTION zadnjiPK
                RETURN NUMBER IS
                BEGIN
                        RETURN nalog_seq.nextval;
        END;
        PROCEDURE upisNaloga
        (kvar NUMBER,
        proizvodac NUMBER,
        zaposlenik NUMBER,
        ime nalog.ime_podnositelja%TYPE,
        prezime nalog.prez_podnositelja%TYPE,
        telefon nalog.tel_podnositelja%TYPE,
        serijski nalog.serijski_proizvoda%TYPE,
        garancija nalog.garancija%TYPE) IS
                kljuc NUMBER := zadnjiPK();
        BEGIN
                INSERT INTO nalog
                VALUES(kljuc, kvar, proizvodac,
                zaposlenik, ime, prezime, telefon, serijski,
                current_date, null, null, garancija, null);
                COMMIT;
                DBMS_OUTPUT.PUT_LINE('Nalog '||kljuc||' je kreiran');
        END;
        PROCEDURE zavrsiNalog
        (sifra NUMBER,
        cijenaObrade nalog.cijena%TYPE) IS
        BEGIN
                UPDATE nalog SET
                datum_otpusta = current_date,
```

```
                cijena = cijenaObrade
                WHERE idnalog = sifra;
                COMMIT;
                DBMS_OUTPUT.PUT_LINE('Nalog '||sifra||' je završen');
        END;
END nalozi;
```

U gornjem primjeru je kreirana specifikacija paketa u kojoj su definirane dvije procedure koje su javno dostupne za zvanje. Nakon toga je definirano i tijelo paketa u kojem se nalaze te iste dvije procedure te njihov kod. Uz to, tu se nalazi i funkcija koja je dostupna za korištenje samo unutar paketa, odnosno privatna je. Prikaz paketa je vidljiv na slici 8. Kod korištenja privatnih funkcija važno je za napomenuti da se one moraju nalaziti prije ostalih potprograma koji ih koriste. Iz tog razloga ona se u tijelu paketa nalazi na vrhu.



Slika 8. Prikaz paketa u Oracle SQL Developeru

Funkcija se koristi za dohvat sljedećeg primarnog ključa za tablicu nalog. Ovdje je korišten objekt koji se zove SEQUENCE i čija je zadaća da vodi zapis o tome koji je sljedeći primarni ključ na redu za određenu tablicu. Za njih je samo važno napomenuti da ih se mora redovito održavati, te ako se on kreira da se svi upisi u tu tablicu obavljaju isključivo preko njega kako ne bi dolazilo do grešaka s ponavljajućim primarnim ključevima.[59] Sljedeća procedura se koristi za upis novog naloga, te tako prima potrebne parametre koji su tipa kolona u tablici. Za dohvat primarnog ključa koristi prethodno opisanu funkciju, te zajedno s ulaznim parametrima upisuje u tablicu naredbom *INSERT*. Druga procedura se koristi za završetak naloga, odnosno postavljanje datuma završetka na današnji datum te upis cijene servisa.

```
BEGIN
        NALOZI.UPISNALOGA(2,3,6,'Petar','Peric','385-123-456','GDG42HNCO',1);
        NALOZI.ZAVRSINALOG(444, 500.00);
END;

Nalog 3212 je kreiran
Nalog 444 je završen
```

[59] Tech On The Net (2014) - Oracle/ PLSQL, *SEQUENCES*, URL: http://www.techonthenet.com/oracle/sequences.php, dostupno 14.06.2014.

14. Okidači

Okidači (*eng. Triggers*) su PL/SQL imenovani blokovi koji se spremaju u bazi kao i procedure, funckije i paketi. Njihova specifičnost je u tome da se oni ne pozivaju, nego se *okidaju* na određene događaje unutar baze podataka. Zbog toga ih se može definirati posebno nad dijelom baze podataka, odnosno nad tablicom, pogledom, shemom ili samom bazom podataka. Tako se okidači mogu pokrenuti na sljedeće tipove događaja[60]:

- DML (*eng. database manipulation language*) naredbe – DELETE, UPDATE i INSERT naredbe
- DDL (*eng. database definition language*) naredbe – CREATE, ALTER i DROP naredbe
- operacije nad bazom – SERVERERROR, LOGON, LOGOF, STARTUP, SHUTDOWN

Općenito se DML okidači pokreču na bilo koju DML naredbu nad tablicom, primjerice INSERT okidač nad tablicom *nalog* se pokreće kod umetanja novog retka u tu tablicu. DDL okidači se pokreču kad se pokrene DDL naredba. Operacije nad bazom se pokreću kod događaja na razini baze, a to su pokretanje i gašenje baze, prijava ili odjava korisnika i slično.

14.1. Kreiranje okidača

U ovom radu će se obraditi samo DML okidač, čija osnovna sintaksa za kreiranje je sljedeća:

```
CREATE [OR REPLACE] TRIGGER ime_okidača
{BEFORE|AFTER} događaj_okidanja ON ime_tablice
[FOR EACH ROW]
[FOLLOWS ime_drugog_okidača]
[ENABLE/DISABLE]
[WHEN uvjet]
DECLARE
        Izrazi za deklariranje varijabli
BEGIN
        Izrazi za izvršavanje
EXCEPTION
        Izrazi za obradu iznimaka
END;
```

Početak sintakse je poznat te se koristi *CREATE [OR REPLACE] TRIGGER ime_okidača*, te ona kreira ili zamjenjuje postojeći okidač pod imenom *ime_okidača*. Nakon toga se mora

[60] Oracle (2009) - Oracle Database PL/SQL Language Reference 11g Release 1, *Using Triggers*, URL: http://docs.oracle.com/cd/B28359_01/appdev.111/b28370/triggers.htm#LNPLS2001, dostupno 14.06.2014.

definirati vrijeme, događaj i objekt za koji se okidač veže. BEFORE i AFTER označuju da li se okidač pokreće prije izvođenja naredbe ili nakon što je naredba izvršena. Pod *događaj_okidanja* se navodi DML naredba na koju se okidač okida, dok se pod *ime_tablice* navodi ime na koji je okidač vezan. Da imamo *BEFORE UPDATE ON nalog* to bi značilo da se okidač pokreće prije izvođenja svake UPDATE naredbe na tablici *nalog*. FOR EACH ROW specificira, da li se okidač pokreće samo jednom za DML naredbu ili posebno za svaki red koji je obuhvaćen tom naredbom. Naredba WHEN provjerava zadani *uvjet* koji mora biti istinit kako bi se okidač pokrenuo, naravno taj dio je opcionalan te ako ga nema okidač se uvijek pokreće. U verziji Oracle 11g dodane su još tri nove naredbe kod kreiranja okidača. Tako su tu FOLLOWS, ENABLE i DISABLE. Postoji mogućnost definiranja više okidača nad istom tablicom u istom trenutku, primjerice dva okidača nad istom tablicom koji se okidaju prije umetanja u tu tablicu. U toj situaciji Oracle ne garantira koji će se od njih prvi izvršiti. Zbog toga je uvedena opcionalna naredba FOLLOWS gdje se okidaču *ime_drugi_okidač* koji dolazi drugi na izvršavanje, navede *FOLLOWS ime_prvi_okidač*. Te se tako osigura da će se prvo izvršiti okidač *ime_prvi_okidač*, a nakon njega *ime_drugi_okidač*.

Okidači mogu biti u dva stanja, a to su ENABLED i DISABLED, te se ta stanja mogu tokom svojeg postojanja mijenjati. Naredbom ENABLE se postavlja okidač u ENABLED stanje te on radi kako je to i predviđeno, dok se naredbom DISABLE on postavlja u DISABLED stanje i kao takav se neće pokretati iako su svi uvjeti ispunjeni. Ali ako postoji potreba za time, može se kod kreiranja specificirati u kojem stanu želimo da okidač bude. Ovaj dio je također opcionalan, te ako nije naveden okidač se kreira u ENABLED stanju.

Sve gore navedene naredbe pripadaju u zaglavlje okidača, te ostatak naredbi pripada tijelu okidača. Tijelo okidača se sastoji od standardnih dijelova bloka, a to su deklarativni dio, izvršni dio i dio za obradu iznimaka. Razlika od procedura i funkcija je ta, da deklarativni dio započinje ključnom riječi DECLARE. No okidač u tijelu ima još neke mogućnosti, a to je pristupanje prethodnim i novim vrijednostima retka naredbama :OLD i :NEW. Vrijednostima se pristupa sintaksom *:NEW.ime_stupca*. No za pojedine DML naredbe vrijede neka ograničenja[61]:

[61] Oracle (2009) - Oracle Database PL/SQL Language Reference 11g Release 1, *Accessing Column Values in Row Triggers*, URL: http://docs.oracle.com/cd/B28359_01/appdev.111/b28370/triggers.htm#i1006702, dostupno 14.06.2014.

- Za INSERT naredbu okidač ima pristup samo na :NEW jer prethodnih vrijednosti nema, pa može pristupiti odnosno dohvatiti i promjeniti vrijednost novih vrijednosti koje se upisuju u tablicu.
- Za DELETE naredbu okidač ima pravo samo na :OLD jer se nove ne upisuju, a stare više neće postojati poslije brisanja
- ZA UPDATE naredbu, može se pristupiti i :NEW i :OLD tako da se pomoću starih i novih mogu izvesti neke operacije i promjeniti vrijednosti koje se upisuju u redak.

No ako se kod naredbe DELETE koristi :NEW neće se javiti greška, jer :NEW vrijednosti su sve postavljene na *null* pa će one biti i korištene. Isti princip vrijedi i za naredbu INSERT. U sljedećem primjeru je definiran okidač koji se pokreće prije modificiranja vrijednosti nad tablicom *nalog*. Da bi upisao trajanje, koristi novu varijablu koja se upisuje pomoću *:NEW.trajanje* te oduzima trenutni datum od datuma koji već postoji u tom retku, odnosno *:OLD.datum_zaprimanja*.

```
CREATE OR REPLACE TRIGGER trajanje
BEFORE UPDATE ON nalog
FOR EACH ROW
BEGIN
        :NEW.trajanje := current_date - :OLD.datum_zaprimanja;
END;

DECLARE
        sifra nalog.idnalog%TYPE := 500;
        redak nalog%ROWTYPE;
BEGIN
        SELECT * INTO redak FROM nalog
        WHERE idnalog = sifra;
        DBMS_OUTPUT.PUT_LINE('Trajanje naloga: '||redak.trajanje||' cijena
        '||redak.cijena||' kn');
        nalozi.zavrsinalog(sifra, 300);
        SELECT * INTO redak FROM nalog
        WHERE idnalog = sifra;
        DBMS_OUTPUT.PUT_LINE('Trajanje naloga: '||redak.trajanje||' dana, cijena
        '||redak.cijena||' kn');
END;

Trajanje naloga: 0 cijena 0 kn
Nalog 500 je završen
Trajanje naloga: 135 dana, cijena 300 kn
```

14.2. Promjena i brisanje okidača[62]

Okidač se može promjeniti samo ponovnom deklaracijom, odnosno novim kreiranjem okidača s uključenim OR REPLACE. Za rad s okidačima se koristi i ALTER naredba ali samo za:

- *ALTER TRIGGER ime_okidača ENABLE* – uključivanje okidača
- *ALTER TRIGGER ime_okidača DISABLE* – isključivanje okidača
- *ALTER TRIGGER ime_okidača COMPILE* – kompajliranje okidača

Za brisanje okidača koristi se naredba *DROP TRIGGER ime_okidača*. Ali i ako se obriše tablica na koju je okidač vezan, s njom se briše i okidač.

14.3. Napomene za rad s okidačima

Okidači su veoma korisni objekti u bazi podataka koji omogućavaju automatsko kreiranje podataka u tablicama kod njihovih promjena, zapisivanje *log-a* o tome tko je, što i kada mijenjao, replikaciju tablica, spriječavanje unosa vrijednosti koje se ne slažu s poslovnim pravilima. Neke napomene za razvoj okidača unutar baze podataka[63]:

- ograničiti veličinu okidača, jer postoji ograničenje od 32K. Ako je potrebno, logika iz okidača se može prebaciti u potprogram koji se zatim poziva iz okidača.
- izbjegavati rekurzivne okidače, primjerice ako se okidač okida na BEFORE UPDATE ne smije koristiti UPDATE naredbu unutar sebe jer će tako pozivati sam sebe sve dok mu ne ponestane memorije.
- ako se mijenja okidač pomoću REPLACE naredbe i mijenja se samo tablica na koju se okidač odnosi, javit će se greška da taj okidač već postoji na drugo tablici
- pripaziti na imena okidača, okidači imaju odvojeni prostor imenovanje nego potprogrami i paketi pa se neće javljati greška ako ima isto ime kao neki potprogram. Ali se to ne smatra dobrom praksom, pa se za ime okidača preporuča staviti naziv tablice i prva slova njegovih aktivatora, primjerice AU (*after update*)
- ne koristiti okidače za provjeru akcija i pravila koja se mogu izvršiti u samoj bazi, primjerice provjeru tipova podataka ili ograničenja UNIQUE, NOT NULL i slično.

[62] Oracle (2009) - Oracle Database PL/SQL Language Reference 11g Release 1, *Modifying Triggers*, URL: http://docs.oracle.com/cd/B28359_01/appdev.111/b28370/triggers.htm#LNPLS2008, dostupno 14.06.2014.
[63] Oracle (2009) - Oracle Database PL/SQL Language Reference 11g Release 1, *Guidelines for Designing Triggers*, URL: http://docs.oracle.com/cd/B28359_01/appdev.111/b28370/triggers.htm#LNPLS2002, dostupno 14.06.2014.

15. Dinamički SQL

Dinamički SQL je metodologija programiranja za stvaranje i pokretanje SQL naredbi kod pokretnja koda.[64] Koristi se kada program treba pokretati DDL naredbe ili kada cijela DML naredba nije poznata kod kompajliranja koda. Postoje dva načina pisanja dinamičkog SQL-a:

- nativni dinamički SQL (*eng. native dynamic SQL*)
- DBMS_SQL paket

Nativni dinamički SQL je lakši za čitati i pisati nego isti kod koji koristi DBMS_SQL paket te se brže izvršava, ali za njega je potrebno znati broj i tip ulaznih i izlaznih varijabli koje će se koristiti. Ako te informacije nisu poznate potrebno je koristiti DBMS_SQL paket.

15.1. Nativni dinamički SQL

Nativni dinamički SQL pokreće većinu SQL naredbe pomoću EXECUTE IMMEDIATE naredbe. Dok SELECT naredbe koje vraćaju više redaka, pokreće pomoću OPEN FOR, FETCH i CLOSE naredbi. Primjer dinamičke SQL naredbe je:

```
SELECT ime_zap, prezime_zap FROM zaposlenik WHERE id_zap = :sifra;
```

U SELECT naredbi iznad nije poznat identifikator po kojem se dohvaćaju podatci, pa se za nju koristi vezani argument (*eng. bind argument*) *:sifra* koji se zadaje tako da se proizvoljno odabere njegovo ime te ispred njega stavi dvotočka. EXECUTE IMMEDIATE naredba se pokreće koristeći sljedeću sintaksu[65]:

```
EXECUTE IMMEDIATE dinamički_SQL
[INTO definirana_varijabla1,defnirana_varijabla2, ...]
[USING [IN | OUT | IN OUT] parametar1,parametar2,...]
[{RETURNING | RETURN} polje1,polje2, ... INTO parametar1,parametar2, ...]
```

dinamički_SQL je varijabla koja sadrži valjanu SQL naredbu ili PL/SQL blok. Klauzula INTO omogućava upisivanje rezultata naredbe u definirane varijable. Ovo je samo ako naredba vraća najviše jedan red. Klauzula USING se koristi kada se SQL naredbi proslijeđuju parametri koji se koriste kako bi dinamički formirali SQL naredbu kod svakog pozivanja. Ako IN, OUT i IN OUT vrste parametara nisu specificirane, koristi se IN vrsta parametra. RETURNING INTO se koristi kod DML naredbi UPDATE, INSERT, DELETE da vrati vrijednosti u zadane varijable.

[64] Oracle (2014) - Oracle Database PL/SQL Language Reference 11g Release 2, *PL/SQL Dynamic SQL*, URL: http://docs.oracle.com/cd/E11882_01/appdev.112/e25519/dynamic.htm#LNPLS011, dostupno 14.06.2014.
[65] Rosenzweig B., Rakhimov E. (2009) – *Oracle PL/SQL by Example Fourth Edition*, str.381.

```
DECLARE
        ddl_naredba VARCHAR2(100);
        ime_tablice VARCHAR2(15) := 'proizvodac';
        sql_naredba VARCHAR2(100);
        plsql_blok VARCHAR2(400);
        ime VARCHAR2(30);
        sifra NUMBER := 3;
BEGIN
        ddl_naredba := 'ALTER TABLE '||ime_tablice||' ADD ovlastenje DATE';
        EXECUTE IMMEDIATE ddl_naredba;

        sql_naredba := 'SELECT ime_proizvodac FROM proizvodac WHERE id_proizvodac =
        :sifra';
        EXECUTE IMMEDIATE sql_naredba
        INTO ime USING sifra;
        DBMS_OUTPUT.PUT_LINE('Ime proizvođača: ' || ime);

        sql_naredba_r := 'UPDATE zaposlenik SET prezime_zap = ''Ivankovic'' WHERE id_zap =
        :1 RETURNING prezime_zap INTO :2';
        EXECUTE IMMEDIATE sql_naredba_r USING sifra RETURNING INTO ime;
        DBMS_OUTPUT.PUT_LINE('Novo prezime zaposlenika: ' || ime);

        plsql_blok :='DECLARE ' ||
                        'datum DATE := current_date; ' ||
                'BEGIN ' ||
                        'UPDATE proizvodac SET ovlastenje = datum ' ||
                        'WHERE id_proizvodac = :sifra;' ||
                        'DBMS_OUTPUT.PUT_LINE(''Blok - novi datum: ''||datum); '
                ||
                'END;';
        EXECUTE IMMEDIATE plsql_blok USING sifra;
END;

Ime proizvođača: Hewlett-Packard

Novo prezime zaposlenika: Ivankovic

Blok - novi datum: 16.06.14
```

U gornjem bloku je prikazano nekoliko primjera dinamičkog SQL-a. Prva SQL naredba je DDL naredba koja mijenja tablicu *proizvodac*, odnosno dodaje joj novi stupac. Kod DDL naredbi postoji razlika kod formiranja, a to je da ne podržavaju vezane argumente. Pa se one dinamički mijenjaju pomoću varijabli i operatora ||. Ako se DDL naredbi proslijedi parametar dolazi do greške *ORA-01027: bind variables not allowed for data definition operations*. Općenito vrijedi pravilo da se dinamičkom SQL-u ne mogu proslijeđivati imena objekata sheme kao vezani argumenti, te se zbog toga imena tablica također upisuju pomoću operatora ||. Tako u sljedećem primjeru zbog toga dolazi do greške *ORA-00903: invalid table name.*

```
DECLARE
        broj NUMBER;
BEGIN
        EXECUTE IMMEDIATE 'SELECT COUNT(*) FROM :tablica'
        INTO broj USING 'proizvodac';
END;
```

Druga naredba dobavlja ime iz tablice *proizvodac*, koristeći parametar *sifra*. Naredba EXECUTE IMMEDIATE koristi klauzulu INTO *ime* gdje *ime* predstavlja varijablu u koju se sprema ime dobavljeno pomoću upita. Parametar se proslijeđuje pomoću klauzule USING *sifra* gdje je *sifra* varijabla koja ima spremljenu vrijednost identifikatora tog retka.

Treća naredba je UPDATE naredba koja mijenja prezime zaposlenika, te koristi klauzulu RETURNING kako bi vratila novu vrijednost u varijablu koja je zadana. Sama naredba koristi dva vezana argumenta gdje je jedan ulazni i određuje identifikator retka, dok je drugi varijabla u koju se sprema varijabla vraćana RETURNING naredbom. Kod EXECUTE IMMEDIATE se koristeći USING odredi varijabla za prvi argument, a koristeći RETURNING INTO odredi varijabla u koju se upisuje izlazna vrijednost proslijeđena od od UPDATE naredbe.

Četvrta naredba je zapravo cijeli PL/SQL blok, te se izvršava isto kao i normalna SQL naredba. Prima jedan ulazni parametar te se pokreće EXECUTE IMMEDIATE naredbom koristeći samo USING klauzulu.

Ako je dinamičkom SQL-u potrebno proslijediti *null* vrijednost to se ne može koristeći USING NULL, jer dolazi do greške *PLS-00457: expressions have to be of SQL types*. Naime *null* nije valjani SQL tip. Da se proslijedi *null* vrijednost treba deklarirati varijablu u deklarativnom dijelu te se ona ne smije inicijalizirati. Tako varijabla sadrži vrijednost *null* te se ona šalje koristeći USING klauzulu.

OPEN-FOR, FETCH i CLOSE klauzule koriste se kod dinamičkog SQL-a za naredbe koje vraćaju više redaka. Odnosno koncept iza tih naredbi je dosta sličan kao rad s kursorima. A koristi kursor varijable.

Kursor varijable ili REF CURSOR je tip varijable, koji dinamički veže SELECT naredbu za sebe kod pokretanja. Kursori uvijek pokazuju na isti memorijski prostor, dok kursor varijable mogu pokazivati na različite memorijske prostore. Pa se tako kursori smatraju statičnima jer pokazuju na memorijski prostor vezan uz jedan SELECT upit, a kursor varijable su dinamčke jer nisu strogo vezane za jedan upit.[66] Kod svakog dohvaćanja SELECT upita sa više redaka

[66] Rosenzweig B., Rakhimov E. (2009) – *Oracle PL/SQL by Example Fourth Edition*, str.471.

Oracle otvara memorijski prostor, odnosno kursor da spremi sve potrebne informacije. Da se pristupi tim informacijama, treba se navesti ime memorijskog prostora ili koristiti kursor varijablu koja pokazuje na njega. REF CURSOR je tip podataka u PL/SQL-u gdje REF stoji umjesto referenca (*eng. reference*), a CURSOR je objekt. Sintaksa za deklariranje kursor varijable je sljedeća:

> *TYPE **ime_kursor_tipa** IS REF CURSOR [RETURN tip_podataka];*
> *ime_kursora **ime_kursor_varijable***

Nakon deklaracije tipa kursor varijable još je potrebno deklarirati i kursor varijablu tog tipa.Postoje dvije vrste kursor varijable i to su:

- snažni tip kursor varijable
- slabi tip kursor varijable

Snažni tip kursor varijable je tip kursor varijable koja ima strogo određeni tip podataka koji prima, odnosnos deklariraju se koji retci upita će se dohvaćati. Tako u gore navedenoj sintaksi se taj tip deklarira pomoću RETURN klauzule. No da bi se takav kursor mogao deklarirati, mora se taj tip zapisa deklarirati prije deklaracije kursora. Slabi tip kursor varijable koristi istu sintaksu samo bez RETURN klauzulue, te tako omogućava prihvat redaka s proizvoljnim stupcima.[67] Za razliku od kursora s kojima se radi kroz četiri koraka, kursor varijable imaju tri koraka potrebna za rad i ta tri koraka su:

- definicija i deklaracija kursor varijable – kursor varijabla se povezuje s SELECT naredbom, izvršava naredbu te definira skup podataka kojem se pristupa preko kursor varijable. Može se više puta otvarati za različite upita, ali kada se otvori za drugu SELECT naredbu više se ne može pristupiti skupu podataka koji je dohvaćen prvom SELECT naredbom. Dinamički SQL je ostvaren ovdje pomoću OPEN FOR klauzule koja ima dodatnu opcionalnu klauzulu USING pomoću koje se popune vezani argumenti SELECT naredbe.
- dohvat redaka – dohvaća redak iz skupa rezultata isto kao i kod kursora FETCH klauzulom, gdje se mora pripaziti da varijabla u koju se prosljeđuje ima isti broj stupaca i iste tipove podataka koje imaju i retci aktivnog skupa. Ako dođe do neslaganja, kod snažnih tipova kursor varijabli dolazi do greške kod kompajliranja dok kod slabih tipova kursor varijabli do greške dolazi kod izvođenja.

[67] Rosenzweig B., Rakhimov E. (2009) – *Oracle PL/SQL by Example Fourth Edition*, str.472.

- zatvaranje kursora – kursor varijabla se ne mora zatvoriti CLOSE klauzulom, nakon svakog dohvata za različitu varijablu, ali se smatra dobrom praksom da se zatvori svaki puta kada se retci aktivnog skupa više neće koristiti i kada se kursor varijabla koristi za sljedeću SELECT naredbu.

```
DECLARE
        TYPE cv_tip IS REF CURSOR;
        cv_zaposlenik cv_tip;
        radnik zaposlenik%ROWTYPE;
BEGIN
        OPEN cv_zaposlenik FOR
                'SELECT * FROM zaposlenik z WHERE z.servis_id_servisa = :sifra'
                USING 1;
        LOOP
                FETCH cv_zaposlenik INTO radnik;
                EXIT WHEN cv_zaposlenik%NOTFOUND;
                DBMS_OUTPUT.PUT_LINE('Servis 1: '||radnik.ime_zap||'
                '||radnik.prezime_zap);
        END LOOP;
        CLOSE cv_zaposlenik;
        OPEN cv_zaposlenik FOR
                'SELECT * FROM zaposlenik z WHERE z.servis_id_servisa = :sifra'
                USING 2;
        LOOP
                FETCH cv_zaposlenik INTO radnik;
                EXIT WHEN cv_zaposlenik%NOTFOUND;
                DBMS_OUTPUT.PUT_LINE('Servis 2: '||radnik.ime_zap||'
                '||radnik.prezime_zap);
        END LOOP;
        CLOSE cv_zaposlenik;
END;
```

```
Servis 1: Ivica Ivic
Servis 1: Marija Maric
Servis 1: Marko Markic
Servis 2: Kruno Krunic
Servis 2: Matija Matic
Servis 2: Kreso Kresic
```

U gornjem primjeru je deklarirana jedna kursor varijabla koja kroz izvršni dio prima dvije različite SELECT naredbe. Kursor varijabla prvo prima SELECT naredbu kojoj je kao vezani argument poslan identifikator servisa u tablici *zaposlenici* te se kursor varijabla veže uz memorijski prostor gdje se nalaze ti retci. Druga SELECT naredba je ista samo je drugi identifikator, te sada kursor varijabla pokazuje na drugo memorijsko mjesto gdje se nalaze drugi retci.

15.2. DBMS_SQL paket

DBMS_SQL paket je sučelje koji koristi dinamički SQL za parsiranje DML i DDL naredbi koristeći PL/SQL. Koristi se u slučajevima kada nije poznata lista stupaca koji se dohvaćaju SELECT naredbom ili kada se moraju vezati mjesta SELECT ili DML izjave.[68] Korištene dinamičkog SQL-a pomoću DBMS_SQL paketa je prikazano u donjem primjeru[69].

```
DECLARE
        sql_string VARCHAR2(200) := 'SELECT ime_zap, prezime_zap,
        placa_zap FROM zaposlenik WHERE servis_id_servisa = :sifra';

        broj_kursora INTEGER;
        radnik zaposlenik%ROWTYPE;
        broj_obradenih INTEGER;
BEGIN
        broj_kursora := DBMS_SQL.OPEN_CURSOR;
        DBMS_OUTPUT.PUT_LINE('Broj kursora: '||broj_kursora);
        DBMS_SQL.PARSE(broj_kursora, sql_string, DBMS_SQL.NATIVE);

        DBMS_SQL.DEFINE_COLUMN(broj_kursora, 1, radnik.ime_zap, 30);
        DBMS_SQL.DEFINE_COLUMN(broj_kursora, 2, radnik.prezime_zap, 30);
        DBMS_SQL.DEFINE_COLUMN(broj_kursora, 3, radnik.placa_zap);

        DBMS_SQL.BIND_VARIABLE(broj_kursora,'sifra', 2);

        broj_obradenih := DBMS_SQL.EXECUTE(broj_kursora);

        LOOP
                IF DBMS_SQL.FETCH_ROWS(broj_kursora) > 0 THEN
                        DBMS_SQL.COLUMN_VALUE(broj_kursora, 1, radnik.ime_zap);
                        DBMS_SQL.COLUMN_VALUE(broj_kursora, 2, radnik.prezime_zap);
                        DBMS_SQL.COLUMN_VALUE(broj_kursora, 3, radnik.placa_zap);

                        DBMS_OUTPUT.PUT_LINE(radnik.ime_zap||'
                        '||radnik.prezime_zap||', '||radnik.placa_zap);
                ELSE
                        EXIT;
                END IF;
        END LOOP;

        DBMS_SQL.CLOSE_CURSOR(broj_kursora);
END;

Broj kursora: 303055365
Kruno Krunic, 7968,43
Matija Matic, 6260,91
```

[68] Oracle (2014) - Oracle Database PL/SQL Language Reference 11g Release 2, *DBMS_SQL Package*, URL: http://docs.oracle.com/cd/E11882_01/appdev.112/e25519/dynamic.htm#LNPLS01108, dostupno 14.06.2014.
[69] Infinite Skills - Video Training (2012) – *Dynamic SQL*, URL:https://www.youtube.com/watch?v=laVDk_v52Vk, dostupno 14.06.2014.

U gornjem primjeru varijabla *sql_string* sadrži SELECT naredbu koja iz tablice *zaposlenik* dohvaća ime, prezime i plaću za zaposlenike određenog servisa te se servis određuje preko vezanog argumenta. DBMS_SQL koristi identifikator kursora koji je INTEGER tipa podataka, te zbog toga varijabla *broj_kursora* predstavlja taj kursor odnosno njegov identifikator.. U varijablu *radnik* koja je tipa retka tablice *zaposlenik* se upisuju dobavljeni retci, a u varijablu *broj_obradenih* se sprema broj dohvaćenih redaka.

Naredbom DBMS_SQL.OPEN_CURSOR se otvara kursor te njegov identifikator koji je u ovom slučaju 303055365, upisuje u varijablu *broj_kursora*. Tako možemo imati više kursora pokrenutih samo preka tog identifikatora. Naredbom DBMS_SQL.PARSE se parsira SELECT naredba sadržana u *sql_string*, odnosno provjeri se sintaksa naredbe. Nakon toga se poveže s našim kursorom preko njegovog identifikatora. Koristeći DBMS_SQL. DEFINE_COLUMN se identificiraju kolone koje su navedene u SELECT naredbi. Naredbom DBMS_SQL.BIND_VARIABLE se vezani argument upita povezuje s varijablom koju mu zadamo. U ovom slučaju to je cijeli broj 2, koji predstavlja identifikator servisa u kojem rade zaposlenici koji se dohvaćaju.

Nakon svega toga se naredbom DBMS_SQL.EXECUTE izvrši kursor te vrati broj redaka koji su dohvaćeni u varijablu *broj_obradenih*. Sada se u kursoru nalaze dohvaćeni retci te ih treba ispisati. Ispisuju se kroz osnovnu petlju gdje se koristi naredba DBMS_SQL.FETCH_ROWS kojoj se proslijeđuje identifikator kursora te ona vraća retke iz kursora sve dok ih ima u kursoru. Ako nema više redaka za dohvat vrati se 0, što predstavlja da je dohvatila 0 redaka te da je prošla kroz sve.

Ako postoji potreba za izmjenu između DBMS_SQL i nativnog dinamičkog SQL-a postoje funkcije koje to omogućavaju:

- DBMS_SQL.TO_REFCURSOR – identifikator kursora pretvara u slabi tip kursor varijable. Potrebno je deklarirati tip kursor varijable i kursor varijablu, te nakon toga u tu varijablu spremiti izlaz funkcije kojoj se proslijedi identifikator kursora.
- DBMS_SQL.TO_CURSOR_NUMBER – pretvara kursor varijablu u identifikator kursora. U varijablu tipa INTEGER se sprema izlaz funkcije kojoj se proslijeđuje kursor varijabla. Prije proslijeđivanja potrebno je otvoriti kursor varijablu i nakon toga nativni SQL joj ne može više pristupati.

16. Objektno orijentirano programiranje u PL/SQL-u[70]

PL/SQL omogućava definiraje objektnog tipa, koji pomaže dizajniranju objektno orijentirane Oracle baze podataka. Objekt se sastoji od atributa i metoda, gdje se atributi koriste za zapis stanja u kojem se objekt nalazi, a metode se koriste za promjenu stanja objekta.[71] Objekt se kreira na razini sheme baze podataka, te se ne može kreirati unutar PL/SQL koda. Kada se tip objekta kreira na razini baze podataka, PL/SQL kod ga može koristiti. Sintaksa za kreiranje objektnog tipa je sljedeća[72]:

```
CREATE [OR REPLACE] TYPE ime_tipa AS OBJECT (
        ime_atributa_1 tip_atributa,
        ime_atributa_2 tip_atributa,
        ...
        ime_atributa_N tip_atributa,

        [specifikacija_metode_1],
        [specifikacija_metode_2],
        ...
        [specifikacija_metode_N]);

[CREATE [OR REPLACE] TYPE BODY ime_tipa AS
        tijelo_metode_1;
        tijelo_metode_2;
        ...
        tijelo_metode_N;]
END;
```

Kreiranje objektnog tipa ima dva koraka, odnosno specifikacija objektno tipa i tijelo objektnog tipa. U specifikaciji objektnog tipa se navode atributi i metode objekta. Atributima se mora odrediti naziv i valjani tip podataka, gdje tip može biti skalarni (NUMBER, VARCHAR2 itd), kolekcija, zapis ili čak drugi objektni tip. Za metode je potrebno samo njezino zaglavlje. Ali kao što se vidi iz sintakse metode nisu obavezni dio objekta. Objekt se može sastojati samo od atributa koji su dovoljni za zapis stanja objekta. Ako objekt nema metoda drugi korak nije potreban. Drugi korak, odnosno tijelo objekta se kreira samo u slučaju da objekt ima metode, te se unutar njega navodi cijela metoda odnosno funkcija ili procedura.

```
CREATE OR REPLACE TYPE radnik AS OBJECT
        (ime VARCHAR2(30),
        prezime VARCHAR2(30),
        placa NUMBER(7,2),
```

[70] Rosenzweig B., Rakhimov E. (2009) – *Oracle PL/SQL by Example Fourth Edition*, str.514.
[71] TutorialsPoint (2014) - PL/SQL Tutorial, *Object Oriented*, URL: http://www.tutorialspoint.com/plsql/plsql_object_oriented.htm, dostupno 14.06.2014
[72] Rosenzweig B., Rakhimov E. (2009) – *Oracle PL/SQL by Example Fourth Edition*, str.551.

```
    adresa VARCHAR2(30),
    grad VARCHAR2(30));
```

Kreiran je objektni tip *radnik* na razini sheme baze podataka, koji sadrži pet atributa kojima se zapisuje stanje objekta. Sam objekt se sada može koristiti u PL/SQL kodu.

```
DECLARE
    radnik_1 radnik;
BEGIN
    SELECT radnik(z.ime_zap, z.prezime_zap, z.placa_zap, s.adresa_servisa,
    s.grad_servisa)
    INTO radnik_1
    FROM zaposlenik z
    JOIN servis s on s.id_servisa = z.servis_id_servisa
    WHERE z.id_zap = 6;

    DBMS_OUTPUT.PUT_LINE('Ime: '||radnik_1.ime);
    DBMS_OUTPUT.PUT_LINE('Prezime: '||radnik_1.prezime);
    DBMS_OUTPUT.PUT_LINE('Placa: '||radnik_1.placa);
    DBMS_OUTPUT.PUT_LINE('Adresa: '||radnik_1.adresa);
    DBMS_OUTPUT.PUT_LINE('Grad: '||radnik_1.grad);
END;

Ime: Kreso
Prezime: Kresic
Placa: 6260,91
Adresa: Radnička cesta 12
Grad: Zagreb
```

U primjeru je deklarirana varijabla *radnik_1* objektnog tipa *radnik*. U izvršnom dijelu PL/SQL bloka se pomoću SELECT naredbe upisuje vrijednost u tu varijablu. Unutar SELECT naredbe je korišten konstruktor objekta *radnik(z.ime_zap, z.prezime_zap, z.placa_zap, s.adresa_servisa, s.grad_servisa)* kako bi se dobivene vrijednosti upisale u varijablu objektnog tipa *radnik*.

16.1. Metode objektnog tipa[73]

Metode objektnog tipa su funkcije i procedure koje specificiraju koje akcije se mogu obavljati nad atributima objekta. One upravljaju ponašanjem i stanjima objekta, te se mogu podjeliti na:

- konstruktor metode (*eng. Constructor Methods*)
- metode atributa (*eng. Member Methods*)
- statične metode (*eng. Static Methods*)

[73] Rosenzweig B., Rakhimov E. (2009) – *Oracle PL/SQL by Example Fourth Edition*, str.531.

Konstruktor metoda je zadana metoda koja ima isto ime kao i objekt. Služi za inicijalizaciju atributa objekta, te prima ulazne parametre koji odgovaraju atributima objekta. No mogu se i definirati vlastiti konstruktori koji ne primaju sve atribute koji se nalaze u objektu, te se tako samo oni postave na zadanu vrijednost dok se ostali postave na *null*. Sintaksa je sljedeća:

```
CONSTRUCTOR FUNCTION ime_isto_kao_tip_objekta
(SELF IN OUT NOCOPY ime_isto_kao_tip_objekta, parametar_1 tip_parametra_1, .., ,
parametar_N tip_parametra_N)
RETURN SELF AS RESULT
```

Konstruktor metoda koristi SELF parametar kao IN OUT parametar koji vraća instancu tog tipa podatka. Sljedeće se koristi NOCOPY klauzula koja se koristi s OUT i IN OUT parametrima. IN OUT i OUT parametri se prenose pomoću vrijednosti, te se u potprogram te vrijednosti kopiraju što kod složenih tipova podataka kao kolekcije, zapisi i objekti može uzrokovati veliko popunjavanje memorije. No NOCOPY naredba isključuje korak kopiranja te vrijednosti proslijeđuje referencom.

Metode atributa su metode koje pružaju pristup atributima instance objekta. Vrijednostima atributa objekta se pristupa pomoću SELF parametra. Sintaksa je sljedeća:

```
MEMBER PROCEDURE ime_metode
(parametar_1 tip_parametra_1, .., , parametar_N tip_parametra_N)
```

Statične metode su metode koje nemaju pristup atributima instance objekta, nego se kreiraju za akcije koje su globalne za cijeli objekt. Za korištenje STATIC metode nije potrebno instancirati objekt, nego se izvodi nad samim objektom. Zato jer nemaju pravo pristupu atributima instance objekta, ne mogu ni koristiti SELF parametar. Sintaksa je sljedeća:

```
STATIC PROCEDURE ime_metode
(parametar_1 tip_parametra_1, .., , parametar_N tip_parametra_N)
```

Primjer kreiranje novog objektnog tipa, odnosno promjena starog tip objekta *radnik* s novom, promjenjenom verzijom. Atributi su ostali isti, ali su dodane tri metode.

```
CREATE OR REPLACE TYPE radnik AS OBJECT
    (ime VARCHAR2(30),
    prezime VARCHAR2(30),
    placa NUMBER(7,2),
    adresa VARCHAR2(30),
    grad VARCHAR2(30),

    CONSTRUCTOR FUNCTION radnik
    (SELF IN OUT NOCOPY radnik, ime VARCHAR2, prezime VARCHAR2)
    RETURN SELF AS RESULT,

    MEMBER PROCEDURE dobaviImePrezime
```

```
        (o_ime OUT VARCHAR2, o_prezime OUT VARCHAR2),

        STATIC PROCEDURE prosjekPlaca
        (prosjek OUT NUMBER));

CREATE OR REPLACE TYPE BODY radnik AS
        CONSTRUCTOR FUNCTION radnik
        (SELF IN OUT NOCOPY radnik, ime VARCHAR2, prezime VARCHAR2)
        RETURN SELF AS RESULT IS
        BEGIN
                SELF.ime := ime;
                SELF.prezime := prezime;
                RETURN;
        END;

        MEMBER PROCEDURE dobaviImePrezime
        (o_ime OUT VARCHAR2, o_prezime OUT VARCHAR2) IS
        BEGIN
                o_ime := SELF.ime;
                o_prezime := SELF.prezime;
        END;

        STATIC PROCEDURE prosjekPlaca
        (prosjek OUT NUMBER) IS
        sql_string VARCHAR2(100);
        BEGIN
                sql_string := 'SELECT AVG(placa_zap) FROM zaposlenik';
                EXECUTE IMMEDIATE sql_string INTO prosjek;
        END;
END;
```

Budući da objektni tip sadrži i metode, potrebno je kreirati i tijelo objektnog tipa gdje se specificiraju metode.Konstruktor metoda omogućava instanciranje samo dva atributa objekta. Metoda atributa vraća vrijednosti atributa *ime* i *prezime* objekta. I zadnja metoda je statična metoda koja vraća prosječnu plaću svih zaposlenika.

```
DECLARE
        radnik_2 radnik;
        prosjek NUMBER := 0;
        ime VARCHAR2(30);
        prezime VARCHAR2(30);
BEGIN
        radnik.prosjekPlaca(prosjek);
        DBMS_OUTPUT.PUT_LINE('Prosjek plaća je: '||prosjek);
        radnik_2 := radnik('Josip','Horvat');
        radnik_2.dobaviImePrezime(ime, prezime);
        DBMS_OUTPUT.PUT_LINE('Ime i prezime: '||ime||' '||prezime);
END;

Prosjek plaća je: 6528,031
Ime i prezime: Josip Horvat
```

16.2. Usporedba objekata[74]

Skalarni tipovi podataka kao NUMBER i DATE se mogu jednostavno uspređivati koristeći operatore usporedbe, ali kod objektnih tipova potrebno je podesiti atribut po kojem želimo da se on uspoređuje. To se radi koristeći jednu od dvije metode atributa:

- metoda mapiranja (*eng. Map Methods*)
- metode poretka (*eng. Order Methods*)

Metoda mapiranja je opcionalna metoda atributa koja specificira koji atribut objektnog tipa se uspoređuje u slučaju uspoređivanja dviju instanci istog objektnog tipa. Metoda ne prima parametre i vraća tip podataka atributa koji se uspoređuje.

```
CREATE OR REPLACE TYPE broj AS OBJECT
        (vrijednost NUMBER,
        vrijednost_2 NUMBER,
        MAP MEMBER FUNCTION usporedbaBroja RETURN NUMBER);

CREATE OR REPLACE TYPE BODY broj AS
        MAP MEMBER FUNCTION usporedbaBroja RETURN NUMBER
        IS BEGIN
                RETURN (vrijednost);
        END;
END;

DECLARE
        broj_1 broj;
        broj_2 broj;
BEGIN
        broj_1 := broj(452, 5);
        broj_2 := broj(174, 1000);
        IF broj_1 > broj_2 THEN
                DBMS_OUTPUT.PUT_LINE('Prva instanca objektnog tipa broj je veća od
                druge instance');
        END IF;
END;
```

Prva instanca objektnog tipa broj je veća od druge instance

U gornjem primjeru je kreiran objektni tip koji sadrži dva atributa i metodu mapiranja koja specificira da se kod usporedbe dviju instance tog objektnog tipa one uspoređuju po atributu *vrijednost*. U primjeru su instancirane dvije instance tog objektnog tipa gdje prva instanca ima atribut *vrijednost* 457, a druga instanca 174. Dok je drugi atribut *vrijednost_2* za prvu instancu 5, a drugu 1000. Kada se uspoređuju te dvije instance vraća se TRUE da je prva instanca veća od druge, jer je 457 veća vrijednost od 174.

[74] Rosenzweig B., Rakhimov E. (2009) – *Oracle PL/SQL by Example Fourth Edition*, str.538.

Metoda poretka ne uspoređuje određeni atribut između dviju instanci objektnog tipa kao metoda mapiranja, nego unutar sebe radi usporedbe nad vrijednostima druge instance objekta. Kao ulaz prima drugu instancu tog objekta te nad svojim vrijednostima i vrijednostima druge instance radi usporedbe te vraća INTEGER tip podataka.

```
CREATE OR REPLACE TYPE broj AS OBJECT
(vrijednost NUMBER,
vrijednost_2 NUMBER,
ORDER MEMBER FUNCTION usporedbaBroja(broj_objekt broj) RETURN INTEGER);

CREATE OR REPLACE TYPE BODY broj AS
ORDER MEMBER FUNCTION usporedbaBroja(broj_objekt broj) RETURN INTEGER
IS BEGIN
        IF vrijednost < broj_objekt.vrijednost THEN RETURN -1;
        ELSIF vrijednost = broj_objekt.vrijednost THEN RETURN 0;
        ELSIF vrijednost > broj_objekt.vrijednost THEN RETURN 1;
        END IF;
        END;
END;

DECLARE
broj_1 broj;
broj_2 broj;
rezultat INTEGER;
BEGIN
        broj_1 := broj(452, 5);
        broj_2 := broj(174, 1000);
        rezultat := broj_1.usporedbaBroja(broj_2);
        DBMS_OUTPUT.PUT_LINE('Rezultat: '||rezultat);
        IF rezultat = 1 THEN
                DBMS_OUTPUT.PUT_LINE('Prva instanca je veća od druge');
        ELSIF rezultat = 0 THEN
                DBMS_OUTPUT.PUT_LINE('Prva instanca je jednaka drugoj');
        ELSIF rezultat = -1 THEN
                DBMS_OUTPUT.PUT_LINE('Druga instanca je veća od prve');
        END IF;
END;

Rezultat: 1
Prva instanca je veća od druge
```

U gornjem primjeru je objektni tip *broj* promijenjen na način da je metoda mapiranja zamjenjena metodom poretka. Sama metoda za parametar prima instancu istog objektnog tipa te uspoređuje njegove vrijednostima sa svojim vrijednostima. Vraća vrijednosti 1, 0 i -1 ovisno o rezultatu usporedbe. A taj rezultat usporedbe se mora u programskom kodu još provjeriti te ovisno o rezultatu izvoditi određeni skup naredbi. Ovakav način usporedbe je primjereniji za složenije usporedbe jer pruža veću mogućnost upravljanja vrijednostima i usporedbe više vrijednosti.

16.3. Nasljeđivanje PL/SQL objekata[75]

U PL/SQL-u je omogućeno da se kod kreiranja novog objekta, naslijedi već postojeći objekt. Kod kreiranja novog objekta važno je da je njegov roditelj objekt deklariran kao NOT FINAL. I sve metode koje će se nadjačati (*eng. override*) moraju biti deklarirane kao NOT FINAL.

```
CREATE OR REPLACE TYPE dvodimenzionalno AS OBJECT(
        duljina NUMBER,
        sirina NUMBER,
        povrsina NUMBER,
        NOT FINAL MEMBER PROCEDURE izracunajPovrsinu) NOT FINAL;

CREATE OR REPLACE TYPE BODY dvodimenzionalno AS
        MEMBER PROCEDURE izracunajPovrsinu IS
        BEGIN
                SELF.povrsina := SELF.duljina * SELF.sirina;
        END;
END;
```

Kod djeteta je važno da se kod kreiranja navede UNDER i ime objekta kojega nasljeđuje. Sada taj objekt može pristupati atributima i metodama objekta roditelja. Uz to može i nadjačati neku njegovu metodu ključnom riječi OVERRIDING te je prilagoditi sebi. Tako je za objekt dvodimenzionalno izračun površine bio po jednoj formuli dok je za objekt trodimenzionalno po drugoj.

```
CREATE OR REPLACE TYPE trodimenzionalno UNDER dvodimenzionalno(
        visina NUMBER,
        OVERRIDING MEMBER PROCEDURE izracunajPovrsinu);

CREATE OR REPLACE TYPE BODY trodimenzionalno AS
        OVERRIDING MEMBER PROCEDURE izracunajPovrsinu IS
        BEGIN
                SELF.povrsina := 2*(SELF.duljina*SELF.sirina + SELF.duljina*SELF.visina
                + SELF.sirina*SELF.visina);
        END;
END;
```

Kod instanciranja objekta važno je paziti na redoslijed atributa. Tako primjerice objekt *dvodimenzionalno* ima duljina, sirina i povrsina te se oni šalju tim redoslijedom. No kod objekta *trodimenzionalno* prvo dolaze atribut koji se nalaze u objektu roditelj, a nakon njih atributi objekta djeteta. Kod poziva metode *izracunajPovrsinu* koristi se metoda objekta djeteta, koja nadjačava metodu objekta roditelj.

[75] TutorialsPoint (2014) - PL/SQL Tutorial, *Object Oriented*, URL: http://www.tutorialspoint.com/plsql/plsql_object_oriented.htm, dostupno 14.06.2014

```
DECLARE
        objekt trodimenzionalno;
BEGIN
        objekt := trodimenzionalno(3,4,null,5);
        objekt.izracunajPovrsinu();
        DBMS_OUTPUT.PUT_LINE('Povrsina: '||objekt.povrsina);
END;

Povrsina: 94
```

16.4. Apstraktni objekti u PL/SQL-u[76]

Apstraktni objekti su objekti koji nemaju svoje instance, te služe isključivo kao roditeljski objekt kod nasljeđivanja. Pomoću njega se kreiraju podtipovi koji imaju neke njegove atribute i metode. Ako se očekuje da se metode apstraktnog tipa nadjačaju za svaki podtip i da svaki podtip ima drugačiji način izvođenja nema smisla definirati metodu u apstraktnom objektu. Apstraktni objekti i njegove metode se deklariraju pomoću NOT INSTANTIABLE NOT FINAL klauzule. Koristi se sljedeća sintaksa:

```
CREATE OR REPLACE TYPE ime_tipa_objekta AS OBJECT(
        atribut_1 tip_podataka_1,
        atribut_N tip_podataka_N,
        NOT INSTANTIABLE NOT FINAL MEMBER PROCEDURE ime_metode)
        NOT INSTANTIABLE NOT FINAL
```

[76] Oracle (2009) - Oracle Database PL/SQL Language Reference 11g Release 1, *Declaring Types and Methods NOT INSTANTIABLE*, URL: http://docs.oracle.com/cd/B28359_01/appdev.111/b28371/adobjbas.htm#CIHJHCJD, dostupno 14.06.2014.

17. Usporedba PL/SQL-a s PL/pgSQL-om

PL/pgSQL (*Procedural Language/PostgreSQL*) je proceduralno proširenje SQL-a za PostgreSQL bazu podataka, te je veoma sličan PL/SQL-u. Osnovna struktura oba jezika je blok, sve varijable moraju biti deklarirane te koristi naredbe za izvršavanje zadataka (*eng. imperative language*).[77] Uza sve sličnosti, postoje sitne razlike u načinu kreiranja potprograma, petlji, načinu pisanja koda i slično. No samo razlike u sintaksi je lako saznati koristeći službenu dokumentaciju, te će se u ovom poglavlju prikazati neke od razlika kod pisanja programa.

Jedna od osnovnih razlika između mogućnosti PL/pgSQL-a i PL/SQL-a je ta da PL/pgSQL omogućava pisanje potprograma u više jezika. Odnosno potprogrami se ne moraju pisati koristeći isključivo PL/pgSQL jezik nego se može koristiti sintaksa jezika s kojim je programer upoznat. PostgreSQL u svojoj standardnoj distribuciji nudi podršku za četiri jezika na kojima se mogu pisati potprogrami, a to su PL/pgSQL, PL/Tcl, PL/Perl, PL/Python.[78] Uz četiri jezika koje podržava sam PostgreSQL, postoji još nekoliko njih koje razvijaju i održavaju sami korisnici. Neki od tih jezika su PL/Java, PL/PHP, PL/Py, PL/R, PL/Ruby, PL/Scheme i PL/sh. [79]

Funkcija potprograma u PL/pgSQL-u se mora pisati kao doslovan tekstualni zapis (*eng. String*), odnosno tijelo same funkcije mora biti između jednostrukih navodnika. U primjeru ispod se vidi da navodnici počinju nakon ključne riječi AS i prije ključne riječi LANGUAGE kojom se definira jezik u kojem je funkcija napisana.

```
CREATE FUNCTION ime_funkcije (argumenti) RETURNS tip_podataka AS '
DECLARE
        deklaracija varijabli
BEGIN
        naredbe
END;'
LANGUAGE 'plpgsql';
```

No kod ovakvog načina pisanja se mora paziti kod daljnjeg korištenja jednostrukih navodnika unutar funkcije. Kod korištenja za potrebe funkcije, primjerice za postavljanje tekstualne varijable one moraju doći u parovima, odnosno *ime_varijable := ''Tekst u varijabli''*. Ako se

[77] PostgreSQL(2014) - PostgreSQL 9.2.8 Documentation, *Porting from Oracle PL/SQL*, URL: http://www.postgresql.org/docs/9.2/static/plpgsql-porting.html, dostupno 23.06.2014.
[78] PostgreSQL(2014) - PostgreSQL 9.1.13 Documentation, *Procedural Languages*, URL: http://www.postgresql.org/docs/9.1/static/xplang.html, dostupno 23.06.2014
[79] PostgreSQL(2014) - PostgreSQL 8.3.23 Documentation, *Procedural Languages*, URL: http://www.postgresql.org/docs/8.3/static/external-pl.html, dostupno 23.06.2014

jednostruki navodnik treba koristiti unutar vrijednosti varijable, onda se taj jednostruki navodnik mora maskirati koristeći obrnutu kosu crtu (*eng. backslash*), odnosno *ime_varijable := ''That\'s''*. [80] Nofunkcija se može pisati i između dva znaka $ umjesto jednostrukih navodnika. Mogu se koristiti sami $$ ili $*ime*$, gdje se znakovi s imenom koriste za raspoznavanje ugnježđenih blokova.[81] Ovakav pristup je prihvatljiviji jer nije potrebno koristiti jednostruke navodnike u parovima niti ih maskirati.

PL/pgSQL nema koncept paketa kao PL/SQL, nego se kod njega koristi shema za grupiranje potprograma.

FOR petlja sa SELECT naredbama radi drugačije. U petlji *FOR varijabla IN select_naredba* kod PL/SQL-a se *varijabla* kreira implicitno te se koristi kroz petlju. Dok kod PL/pgSQL-a se mora deklarirati varijabla točno tog tipa koji se vraća SELECT naredbom te se kao takva koristi u petlji. Prednost je da se može pristupiti vrijednostima te varijable i kada se izađe iz petlje.[82]

FOR petlja koja ide unatrag, odnosno REVERSE također radi drugačije za vrijednosti *vrijednost1..vrijednost2*, jer se kod PL/SQL-a odbrojava od *vrijednost2* do *vrijednost1*. A kod PL/pgSQL-a od *vrijednost1* do *vrijednost2*.

PL/pgSQL funkcije se ponašaju kao jedna transakcija, odnosno unutar njih se ne mogu započinjati i završavati transakcije. Ili se izvrši cijela funkcija ili cijela funkcija propadne.[83]

PL/pgSQL jezik je veoma sličan PL/SQL-u po svim funkcionalnostima. Sama sintaksa je dosta slična, ali postoje sitne razlike. Jedna od većih prednosti PL/pgSQL-a je ta da se potprogrami mogu pisati na više programskih jezika čime se olakšava pisanje za veću grupu ljudi. Dok je prednost PL/SQL-a veća mogućnost rada s transakcijama i rad s paketima koji olakšavaju grupiranje potprograma.

[80] Ewald Geschwinde E, Hans-Jürgen Schönig (2002) - *PostgreSQL Developer's Handbook*, str.149.
[81] PostgreSQL(2014) - PostgreSQL 9.2.8 Documentation, *Handling of Quotation Marks*, URL: http://www.postgresql.org/docs/9.2/static/plpgsql-development-tips.html#PLPGSQL-QUOTE-TIPS, dostupno 23.06.2014.
[82] PostgreSQL(2014) - PostgreSQL 9.2.8 Documentation, *Porting from Oracle PL/SQL*, URL: http://www.postgresql.org/docs/9.2/static/plpgsql-porting.html , dostupno 23.06.2014.
[83] PostgreSQL(2014) - PostgreSQL 9.3.4. Documentation, *Structure of PL/pgSQL*, URL: http://www.postgresql.org/docs/current/interactive/plpgsql-structure.html, dostupno 23.06.2014.

18. Usporedba PL/SQL-a s T-SQL-om[84]

T-SQL (*Transact-SQL*) je proceduralno proširenje SQL-a za koje se koristi za Microsoft SQL Server. Najveća razlika naspram PL/SQL-a se primjećuje u samoj sintaksi. U T-SQL kodu sve varijable se označavaju sa znakom @ ispred naziva i kod deklaracije i kod korištenja. Varijable se deklariraju nakon ključne riječi BEGIN, dok su se kod PL/SQL-a deklarirale prije nje u deklarativnom dijelu. Za postavljanje varijable se koristi ključna riječ SET.

```
CREATE OR REPLACE FUNCTION
kvadrat(ulaz NUMBER)
RETURN NUMBER IS
        izlaz NUMBER;
BEGIN
        izlaz := ulaz * ulaz;
        RETURN izlaz;
END;
```

```
CREATE FUNCTION kvadrat(@ulaz INT)
RETURNS INT AS
BEGIN
        DECLARE @izlaz INT;
        SET @varijabla = @ulaz * @ulaz;
        RETURN @varijabla;
END
```

Sintaksa je podosta različita od PL/SQL-a, ali je intuitvna i jednostavna za korištenje kao i PL/SQL i PL/pgSQL. Za očekivati je da postoje male razlike, ali se kao i u prethodnom poglavlju one neće posebno obraditi nego će se obraditi veće razlike kod pisanja koda.

T-SQL ne podržava usidrene tipove kao PL/SQL, odnosno nema %ROWTYPE i %TYPE usidrene tipove podataka. Za svaku varijablu i parametar se mora koristiti osnovni tip podataka što predstavlja određeni nedostatak kod održavanja funkcija i koda. Kod se mora održavati da ako radi s tablicama baze da koristi tipove podataka kao i u tablicama.

Isto kao i PL/pgSQL u T-SQL-u ne postoji koncept paketa, te jedini način za grupiranje funkcija je koristeći sheme.

T-SQL nema BEFORE okidača za razliku od PL/SQL-a i PL/pgSQL-a. Postoje samo AFTER i INSTEAD OF okidači gdje se INSTEAD OF mogu koristiti umjesto BEFORE okidača. Oni rade na način da kod upisivanja u tablicu, zapišu podatke u virtualnu tablicu, nakon toga obrade podatke ili odrade potrebne akcije. Kada odrade, novo dobivene vrijednosti se zajedno s vrijednostima iz virtualne tablice upisuju u tablicu u koju su trebali biti upisani.[85]

[84] Burleson Consulting (2009) - *Convert SQL Server T-SQL to Oracle PL/SQL,* URL: http://www.dba-oracle.com/t_convent_sql_server_tsql_oracle_plsql.htm, dostupno 23.06.2014.
[85] SQL Authority (2013) - *How to Use Instead of Trigger*, URL:http://blog.sqlauthority.com/2013/01/24/sql-server-how-to-use-instead-of-trigger-guest-post-by-vikas-munjal-koenig-solutions, dostupno 23.06.2014.

19. Zaključak

PL/SQL kao proširenje SQL-a pruža velik broj mogućnosti za manipulaciju podatcima baze podataka. Dok smanjuje broj putovanja po mreži koja bi bila potrebna za svaku pojedinu SQL naredbu, ujedno i smanjuje opterećenje aplikacije zbog premještanja dijela obrade na bazu podataka.

Korištenjem PL/SQL-a se velik dio poslovne logike može prebaciti na bazu podataka, koja danas najčešće služi samo kao podatkovni sloj. Uz korištenjem paketa se mogu grupirati potprogrami koji imaju sličnu namjenu i dalje koristiti kao i ostali potprogrami. Modularnost prisutna u PL/SQL-u omogućava lako održavanje i pisanje koda. Posebice koristeći usidrene tipove podataka, gdje se smanjuje potreba održavanja tipova podataka u potprogramima u slučaju promjene tipa podataka na stupcu tablice.

Približavanje principa objektno orijentiranog programiranja još je jedan korak bliže korištenju baze podataka za obradu podataka. Objektni tipovi s atributima i vlastitim metodama su veoma jednostavni za kreiranje te im je način korištenja isti kao u ostalim objektno orijentiranim jezicima.

Glavne karakteristike PL/SQL-a su obrađene u ovom radu, ali on pruža još zanimljivih i korisnih mogućnosti, naredbi, standardnih paketa i koncepata. Svi oni su veoma detaljno objašnjeni u Oracle dokumentaciji, gdje se mogu naći njihove karakteristike i upute za korištenje.

20. Literatura

- Rosenzweig B., Rakhimov E. (2009) – *Oracle PL/SQL by Example Fourth Edition*, Boston: Pearson Education, Inc.
- Oracle (2005) - *Oracle Database PL/SQL Language Reference 10g Release 2*, URL: http://docs.oracle.com/cd/B19306_01/appdev.102/b14261/toc.htm, dostupno 14.06.2014
- Oracle (2009) - *Oracle Database PL/SQL Language Reference 11g Release 1*, URL: http://docs.oracle.com/cd/B28359_01/appdev.111/b28370/toc.htm, dostupno 14.06.2014.
- Oracle (2014) - *Oracle Database PL/SQL Language Reference 11g Release 2*, URL: http://docs.oracle.com/cd/E11882_01/appdev.112/e25519/toc.htm, dostupno 14.06.2014.
- Tech On The Net (2014) - *Oracle/ PLSQL*, URL: http://www.techonthenet.com/oracle/index.php, dostupno 14.06.2014.
- TutorialsPoint (2014) - *PL/SQL Tutorial*, URL: http://www.tutorialspoint.com/plsql/plsql_tutorial.pdf, dostupno 14.06.2014.
- ZenTut (2014) - *PL/SQL*, URL: http://www.zentut.com/plsql-tutorial /, dostupno 14.06.2014.
- Infinite Skills - Video Training (2012) – *Dynamic SQL*, URL:https://www.youtube.com/watch?v=laVDk_v52Vk, dostupno 14.06.2014.
- Ewald Geschwinde, Hans-Jürgen Schönig (2002) - *PostgreSQL Developer's Handbook*, Sams Publishing
- PostgreSQL(2014) - *PostgreSQL 8.3.23 Documentation*, URL: http://www.postgresql.org/docs/8.3/static/index.html, dostupno 23.06.2014.
- PostgreSQL(2014) - *PostgreSQL 9.1.13 Documentation*, URL: http://www.postgresql.org/docs/9.1/static/index.html, dostupno 23.06.2014.
- PostgreSQL(2014) - *PostgreSQL 9.2.8 Documentation*, URL: http://www.postgresql.org/docs/9.2/static/index.html, dostupno 23.06.2014.
- PostgreSQL(2014) - *PostgreSQL 9.3.4. Documentation*, URL: http://www.postgresql.org/docs/9.3/static/index.html, dostupno 23.06.2014.

- SQL Authority (2013) - *How to Use Instead of Trigger*, URL:http://blog.sqlauthority.com/2013/01/24/sql-server-how-to-use-instead-of-trigger-guest-post-by-vikas-munjal-koenig-solutions, dostupno 23.06.2014.
- Burleson-Consulting (2014) – *Oracle PL/SQL Tuning Tips*, URL: http://www.dba-oracle.com/plsql, dostupno 23.06.2014.