Oracle PL/SQL jezik - priručnik kroz praktične primjere

Beljo, Bruno

Sveučilište u Rijeci – Odjel za informatiku

Jednopredmetna informatika

Bruno Beljo

Oracle PL/SQL jezik - priručnik kroz praktične primjere

Završni rad

Mentor: izv. prof. dr. sc. Patrizia Poščić

Rijeka, 15.09.2018.

Sadržaj

Sažetak

PL/SQL proceduralni je programski jezik napravljen kao proširenje SQL-a. Dio je mehanizama Oracle baza podataka. Za svakoga tko se bavi aplikacijama temeljenima na bazama podataka u klijent/server sustavu, iznimno je korisno znati se služiti ovim programskim jezikom. Radi se o moćnom programskom alatu za rad s bazama podataka s kojim je moguće puno više postići nego samo s SQL-om, programerima omogućuje puno veću efikasnost, a krajnjem korisniku iznimno pojednostavljuje rad s bazama. Ovaj je rad zamišljen kao priručnik kroz sintaksu i mogućnosti PL/SQL-a s praktičnim primjerima. U prvom je poglavlju predstavljen kratki uvod u PL/SQL, s naglaskom na neke od njegovih osnovnih karakteristika i na osnovnu sintaksu. U drugom poglavlju najprije je predstavljen poslovni slučaj, odnosno ogledna baza podataka (diskografske kuće) nad kojom je izvršena većina primjera, a zatim su kroz praktične primjere predstavljene neke od naprednijih mogućnosti te dijelova sintakse PL/SQL-a, kao što su kursori, petlje, polja i slično.

Ključne riječi: PL/SQL, SQL, programski jezik, Oracle, baze podataka, varijable, kursori, petlje, polja, procedure, funkcije, paketi

1. Uvod

1.1. Ukratko o PL/SQL-u

PL/SQL je proceduralni programski jezik za strukturirani jezik upita (*Procedural Language /Sequencial Query Language*) s mnogo namjena. Pripada mehanizmima Oracle baze podataka, a osnovna svrha mu je dati pozadinu procedure u kojoj su konstruirane SQL naredbe [1].

Kompanije Oracle razvila je programski jezik PL/SQL tijekom 1980.-ih, kao proceduralno proširenje SQL jezika za SQL Oracle relacijske baze [3].

PL/SQL omogućuje korištenje i pohranu proceduralnog koda i SQL izjava unutar baze podataka. Pomoću ovog programskog jezika omogućena je integracija SQL-a i koncepata uobičajenih za programiranje, kao što su varijable, procesuiranje po uvjetima (IF-THEN-ELSE strukture), jednostavne petlje (FOR i WHILE), te zaustavljanje pogrešaka. DBMS (*DataBase Management System* – sustav za upravljanje bazama podataka) izvršava proceduralni kod kao cjelinu nakon direktnog ili indirektnog pozivanja od strane konačnog korisnika [6].

PL/SQL može se pozivati direktno iz SQL* Plus sučelja, kao i iz vanjskih programskih jezika prema bazi podataka. Osnovna sintaksa PL/SQL-a utemeljena je na programskim jezicima ADA i Pascal. Osim pri Oracle bazi podataka, PL/SQL dostupan je i u TimesTen in-memory database i IBM DB 2 [3].

Korisnici pomoću programskog jezika PL/SQL mogu, između ostalog, kreirati [6]**:**

- Neimenovane PL/SQL blokove
- Pohranjene procedure
- PL/SQL funkcije

1.2. Mogućnosti i značajke PL/SQL-a

PL/SQL predstavlja skup proceduralnih proširenja SQL-a koja uspješno spajaju prednosti proceduralnih (PL/SQL) i neproceduralnih programskih jezika (SQL). PL/SQL sastavni je dio mnogih Oracle proizvoda. Ugrađen je i u server baze podataka Oracle gdje omogućuje izradu i upotrebu pohranjenih procedura, funkcija i paketa [5].

Programski jezik PL/SQL ima podršku za korištenje i statičkog i dinamičkog SQL-a [1].

PL/SQL se, kao i SQL, može pisati i pokretati u SQL* Plusu ili SQL Worksheetu. U PL/SQL-u moguće je pokretati sve osnovne SQL naredbe; SELECT, INSERT, UPDATE i DELETE te dodatno i naredbe COMMIT i ROLLBACK (to su SQL izjave za kontrolu transakcija). Uz SQL naredbe, u PL/SQL-u moguće je stvarati varijable te im dodjeljivati vrijednosti, služiti se IF-THEN-ELSE logikom, koristiti LOOP i GOTO strukture, te raditi usporedbe [1].

Možda i najvažnija prednost PL/SQL-a je u tome što omogućuje slanje kompletnog bloka naredbi prema bazi podataka odjednom. Za nekoga tko se bavi razvojem aplikacija temeljenih na bazama podataka u klijent/server sustavu, vrlo je korisno znati se služiti PL/SQL-om. Pomoću PL/SQL-a postižu se bolje performanse aplikacije, kao i čitavog sustava, zato što [5]:

- umjesto interpretiranih SQL izjava koriste se prevedeni programi koji se brže izvršavaju
- značajno se smanjuje mrežni promet između klijenata i servera

Također, PL/SQL i programerima omogućuje visoku produktivnost jer istovremeno mogu izvršavati upite nad bazom te mijenjati i ažurirati podatke u bazi. Sve aplikacije pisane u ovom programskom jeziku potpuno su prenosive [4].

Sigurnost je u PL/SQL-u na iznimno visokoj razini. Omogućen je pristup svim unaprijed definiranim SQL paketima. PL/SQL ima i podršku za objektno orijentirano programiranje, kao i za razvoj web aplikacija [1].

1.3. Osnovna sintaksa PL/SQL-a

PL/SQL kao programski jezik strukturiran je po blokovima, što znači da je svaki kod pisan u PL/SQL-u podijeljen u logičke blokove [4].

Svaki redak koda mora završiti sa znakom; [4].

Ovako izgleda osnovna struktura PL/SQL koda [1]:

```
DECLARE
      declare block
BEGIN
      body block
EXCEPTION
      exception block
END;
```

Svaki logički blok odnosi se na određeni dio PL/SQL programa. U bloku DECLARE potrebno je deklarirati i definirati sve varijable i kursore[1]. Pod BEGIN (u `'body block'`-u) piše se izvršni kod, odnosno sve naredbe koje bi program trebao pokrenuti, dok posljednji blok EXCEPTION služi za rukovanje greškama te se pokreće samo ako dođe do prethodno definirane iznimke[4].Od ova tri bloka, samo je `'body block'` obavezan. Nadalje, on mora obavezno sadržavati barem jednu naredbu koju će program izvršiti. Njega možemo i podijeliti u podblokove[1].

Primjer 1 pokazuje kako izgleda vrlo jednostavni PL/SQL kod. U početnom DECLARE bloku deklarirana je i definirana varijabla 'poruka' znakovnog tipa podataka[2] `varchar2` duljine 30, što znači da u nju možemo pospremiti niz bilo kakvih znakova, kojih ukupno može biti najviše 30, uključujući razmake. U `'body block'`-u koji započinje ključnom riječi BEGIN definirana je izvršna naredba koja će pokrenuti ispis niza znakova pohranjenih

[1]Poglavlje 2.7. – Kursori
[2]Poglavlje 2.2. – Tipovi podataka

u varijabli 'poruka'. Prije svega moramo pokrenuti naredbu SET SERVEROUTPUT ON da bi omogućili ikakav ispis u programu, odnosno normalno funkcioniranje naredbe `dbms_output.put_line(tekstualna_varijabla)`. Ova naredba izvršiti će ispis bilo koje varijable nekog od tekstualnih tipova podataka, a koju joj proslijedimo kao parametar u zagradi. Naredba SET SERVEROUTPUT ON vrijedi dok god smo u istoj sesiji rada u komandnoj liniji. Dakle, jednom kada smo ju pokrenuli, dok god radimo u komandnoj liniji, ispis će nam funkcionirati. (*Napomena: u svakom idućem primjeru koji sadrži bilo kakav ispis podrazumijevamo da je u jednom trenutku prije programa (tijekom trenutne sesije) pokrenuta naredba SET SERVEROUTPUT ON.)

Na kraju svakog programa, nakon ključne riječi END kojom se zaključuje posljednji PL/SQL blok, znakom „/“ označuje se kraj unosa u komandnu liniju i kompajleru daje signal za kraj programa. Sve što se prikaže u komandnoj liniji nakon tog znaka rezultat je programa, odnosno izvršnih naredbi koje su dio programa. Da bi se program pokrenuo, nakon znaka „/“ potrebno je stisnuti tipku Enter.

Razlikujemo imenovane i neimenovane PL/SQL blokove [8].

PL/SQL blok u ovom primjeru je neimenovan (nema nikakvo određeno ime) te će se izvršiti trenutno, odmah po pritisku na tipku ENTER. Nakon što se program uspješno izvrši (bez grešaka), u komandnoj liniji vidjet ćemo poruku „PL/SQL procedure successfully completed. [6]“ Neimenovani blokovi trenutno se kompajliraju te se ne mogu naknadno ponovno pozivati jer se oni ne spremaju na server kao objekti baze podataka [8].

```
SET SERVEROUTPUT ON;

DECLARE
      poruka varchar2(30) := 'Hello, World!';
BEGIN
      dbms_output.put_line(poruka);
END;
/

Hello, World!

PL/SQL procedure successfully completed.
```

Primjer 1. Ispis poruke 'Hello, World' u PL/SQL-u

S druge strane, imenovanim PL/SQL blokovima daje se određeno, jedinstveno ime. Za razliku od neimenovanih blokova, oni se pod tim imenom spremaju na server kao objekti baze podataka. Može ih se pozivati više puta, dok god se nalaze na serveru. Također, mogu biti pozvani i iz drugih blokova. Struktura im je ista kao kod neimenovanih blokova, osim što ne počinju ključnom riječi DECLARE već CREATE koja govori kompajleru da ih stvara kao objekte baze podataka. Imenovani blokovi mogu biti ugniježđeni unutar drugih blokova, te mogu sadržavati ugniježđene blokove [8].

Dvije najkorištenije vrste imenovanih PL/SQL blokova su procedure[3] i funkcije[4]. O njima više u pripadnim poglavljima.

Dva su načina na koja možemo pisati komentare unutar koda PL/SQL programa. Kada se radi o samo jednoj liniji komentara, potrebno je komentar odvojiti dvostrukom crticom („--„), kao u primjeru 2. Ako komentar ima više linija, odvajamo ga od ostatka koda kombinacijom znakova "/" i „*", kako je također prikazano u primjeru.

```
DECLARE

      poruka varchar2(10) := 'PORUKA!';

BEGIN

      -- Ovo je komentar!
      /*
      * Ovo je komentar s više linija!
      * Ovo je druga linija komentara!.
      */
      dbms_output.put_line(poruka);

END;
/

PORUKA!

PL/SQL procedure successfully completed.
```

Primjer 2. Komentari u PL/SQL-u

[3] Poglavlje 2.11. – Procedure
[4] Poglavlje 2.12. – Funkcije

2. PL/SQL

2.1. Poslovni slučaj

2.1.1. Model entiteti-veze

U ovom poglavlju predstavljena je relacijska baza podataka diskografske kuće koja je korištena za većinu primjera u drugim poglavljima. Nad bazom su izvedeni neki od primjera korištenja kursora, polja, petlji i ostalih funkcionalnosti PL/SQL-a.

Relacijska baza sastoji se od 5 međusobno povezanih tablica; BEND, ALBUM, PJESMA, SINGL i ČLAN BENDA. Radi se o vrlo jednostavnoj hijerarhiji u kojoj svaka tablica ima svoju ulogu. Tablice su napravljene tako da su pogodne za primjere. Zgodno je, primjerice, ispisati samo 2 ili 3 ključna stupca iz svake tablice.

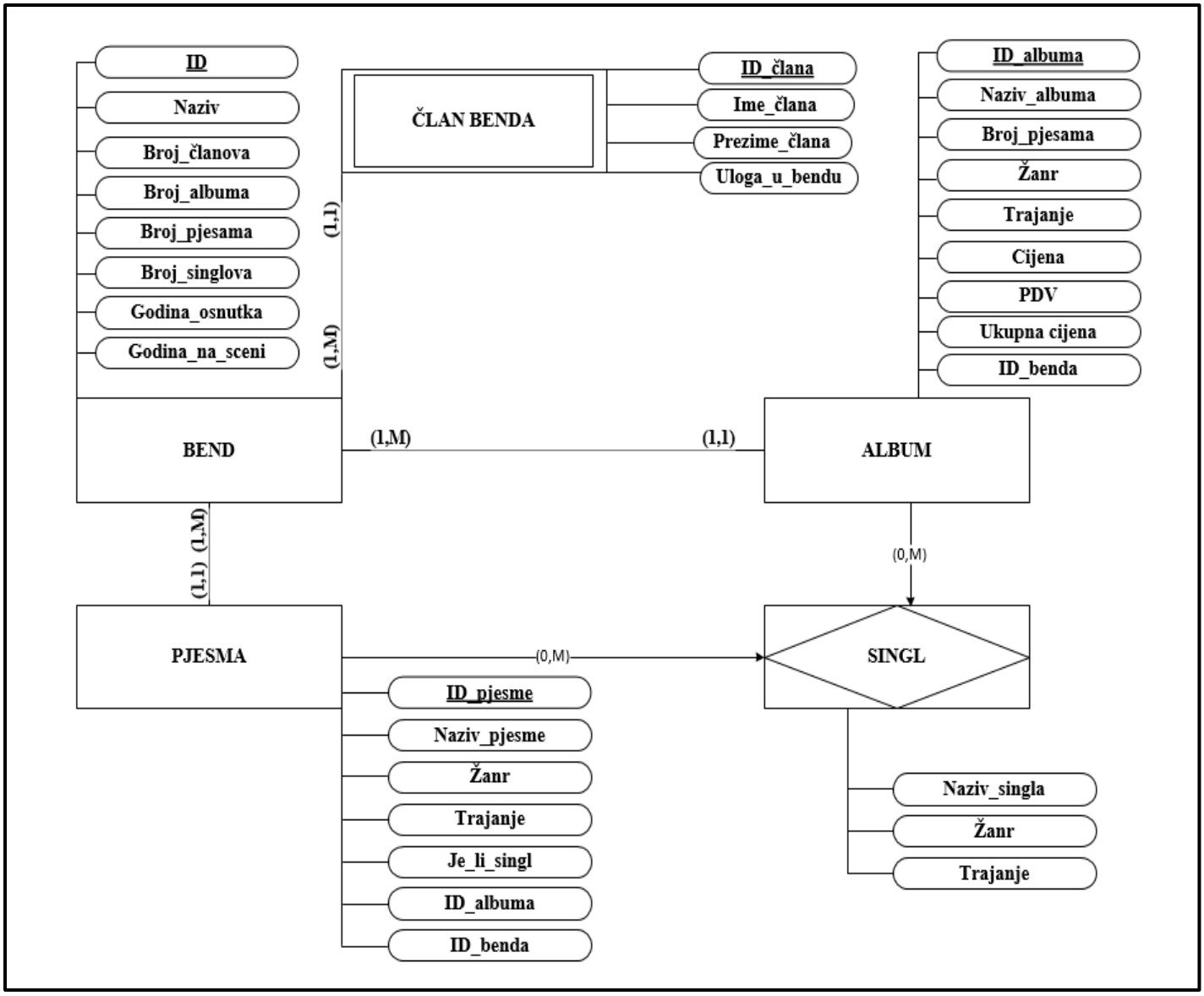
Tablice BEND, ALBUM i PJESMA jednostavne su tablice, što znači da imaju u potpunosti vlastite primarne ključeve. Jedinstvena identifikacija retka u bilo kojoj od ovih tablica ne ovisi o nijednoj drugoj tablici osim nje same. U tablici BEND primarni je ključ stupac 'ID', u tablici ALBUM je to 'ID_albuma', a u tablici PJESMA 'ID_pjesme'.

Tablice SINGL i ČLAN BENDA imaju složene primarne ključeve te jedinstvena identifikacija retka u ovim tablica ovisi i o drugim tablicama. Tablica SINGL agregacija je tablica ALBUM i PJESMA te je njen primarni ključ kompozitni, sastavljen od primarnih ključeva tih dvaju tablica ('ID_albuma' i 'ID_pjesme').

Tablica ČLAN BENDA slabi je entitet tablice BEND, te se njen primarni ključ sastoji od vlastitog dijela ('ID_clana') te primarnog ključa tablice BEND ('ID_benda').

Jedan bend može izdati više albuma. Jedan bend može imati i više članova, ali jedan je član uvijek u samo jednom bendu. Jedan album može imati više pjesama i može biti izdan od strane samo jednog benda. Jedna pjesma može se pojaviti na više albuma a jedan album sadrži više pjesama, zato nastaje spomenuta agregacija, koju predstavlja tablica SINGL.

U nastavku je baza podataka predstavljena kroz model entiteti-veze.



Slika 1. Model entiteti-veze [7]; (Entity-relationship model (ER), Chen, 1976.), MIRIS metodologija

2.1.2. Prikaz baze podataka po tablicama

U ovom su potpoglavlju prikazane konkretne tablice baze podataka u svom originalnom stanju, prije eventualnih izmjena nastalih pri nekim konkretnim primjerima.

Tablica BEND ima 5 zapisa. Svaki je od njih jedinstveno identificiran primarnim ključem koji se sastoji od prvog stupca, 'ID'. Većina ostalih stupaca su numeričkog cjelobrojnog tipa podataka INT, samo je stupac 'Naziv' znakovnog tipa varchar2.

Tablica ALBUM ima 15 zapisa. Za svaki bend iz tablice BEND unesena su po 3 albuma. Svaki je album jedinstveno identificiran primarnim ključem koji se sastoji od stupca 'ID Albuma'. Stupci 'Naziv albuma' i 'Žanr' su znakovnog tipa podataka varchar2, dok su svi ostali stupci numeričkog cjelobrojnog tipa podataka INT.

Tablica PJESMA ima ukupno 45 zapisa. Za svaki album iz tablice ALBUM unesene su po 3 pjesme. Svaki je redak jedinstveno identificiran primarnim ključem koji se sastoji od stupca 'ID pjesme'. Stupci 'Naziv pjesme', 'Je_li_singl', 'Žanr' i 'Trajanje' znakovnog su tipa podataka varchar2, dok su svi ostali stupci numeričkog cjelobrojnog tipa podataka INT.

Tablica SINGL ima 19 zapisa. Tablica je gotovo identična tablici PJESMA, a zamišljena je kao agregacija tablice PJESMA i tablice ALBUM. Ovdje je uneseno nekoliko pjesama iz tablice PJESMA na temelju vrijednosti u stupcu 'je_li_singl'. Sve pjesme iz tablice PJESME koje za taj stupac imaju vrijednost 'DA', ponavljaju se i u tablici SINGL. Ovdje je svaki redak, budući da se radi o agregaciji, jedinstveno identificiran složenim primarnim ključem koji se sastoji od dvaju stupaca, 'ID albuma' i 'ID pjesme' (primarni ključevi tablica ALBUM i PJESMA). Stupci 'Naziv singla', 'Žanr singla' i 'Trajanje singla' znakovnog su tipa podataka varchar2, dok su stupci 'ID Pjesme' i 'ID Albuma' numeričkog cjelobrojnog tipa podataka INT.

Tablica ČLAN BENDA zamišljena je kao slabi entitet tablice BEND. U njoj se nalaze svi individualni članovi svakog pojedinog benda. Ovdje je svaki redak, budući da se radi o slabom tipu entiteta vezanom uz tablicu BEND, jedinstveno identificiran složenim primarnim ključem koji se sastoji od stupca 'ID Člana' specifičnog za ovu tablicu i stupca 'ID Benda' koji je primarni ključ iz tablice BEND. Stupci 'Ime člana', 'Prezime člana' i 'Uloga u bendu' znakovnog su tipa podataka Varchar2, dok su stupci 'ID Člana' i 'ID Benda' numeričkog cjelobrojnog tipa podataka INT. U tablici su ukupno 24 zapisa.

Tablica 1. Bendovi (izvođači)

BEND							
ID	***Naziv***	***Broj članova***	***Broj albuma***	***Broj pjesama***	***Broj singlova***	***Godina osnutka***	***Godina na sceni***
1357	Radiohead	5	9	99	30	1985.	33
2468	U2	4	14	150	67	1976.	42
3579	Arctic Monkeys	4	6	65	23	2002.	16
4680	Crvena Jabuka	5	15	170	62	1985.	33
5791	Opća Opasnost	6	7	75	30	1992.	26

Tablica 2. Albumi

ALBUM								
ID albuma	***Naziv albuma***	***Broj pjesama***	***Žanr***	***Trajanje***	***Cijena***	***PDV***	***Ukupna cijena***	***ID benda***
1011	Pablo Honey	12	Pop rock	42:11	75 kn	19 kn	94 kn	1357
1012	The Bends	12	Indie rock	48:37	70 kn	18 kn	88 kn	1357
1013	OK Computer	12	Alt rock	53:21	80 kn	20 kn	100 kn	1357
2121	Joshua Tree	11	Rock	50:11	100 kn	25 kn	125 kn	2468
2122	Achtung Baby	12	Alt rock	55:27	100 kn	25 kn	125 kn	2468
2123	Pop	12	Alt dance	60:09	100 kn	25 kn	125 kn	2468
3231	Favourite Worst Nightmare	12	Punk rock	37:34	82 kn	20 kn	102 kn	3579
3232	Humbug	10	Alt rock	39:20	91 kn	23 kn	114 kn	3579
3233	AM	10	Rock	41:43	100 kn	25 kn	125 kn	3579
4341	Crvena Jabuka	10	Novi val	37:23	39 kn	10 kn	49 kn	4680
4342	Za sve ove godine	12	Pop rock	38:46	39 kn	10 kn	49 kn	4680
4343	Sanjati	12	Pop rock	46:38	39 kn	10 kn	49 kn	4680
5451	Opća opasnost	9	Rock	34:34	49 kn	12 kn	61 kn	5791
5452	Treba mi nešto jače od sna	10	Rock	38:50	49 kn	12 kn	61 kn	5791
5453	Amerika	13	Rock	54:06	49 kn	12 kn	61 kn	5791

Tablica 3. Pjesme

PJESMA						
ID pjesme	***Naziv pjesme***	***Žanr***	***Trajanje***	***Je li singl***	***ID albuma***	***ID benda***
1151	You	Pop rock	3:29	NE	1011	1357
1162	Creep	Soft rock	3:56	DA	1011	1357
1173	How Do You?	Rock	2:12	NE	1011	1357
1291	High and Dry	Indie rock	4:17	DA	1012	1357
1282	(Nice Dream)	Indie rock	3:53	NE	1012	1357
1273	Just	Indie rock	3:54	NE	1012	1357
1341	Airbag	Alt rock	4:44	NE	1013	1357
1352	Paranoid Android	Alt rock	6:23	NE	1013	1357
1363	Karma Police	Alt rock	4:21	DA	1013	1357
2131	Where the Streets Have No Name	Rock	5:38	DA	2121	2468
2142	I Still Haven't Found What I'm Loking For	Gospel, rock	4:38	DA	2121	2468
2153	With or Without You	Soft rock	4:56	DA	2121	2468
2251	Zoo Station	Alt rock	4:36	NE	2122	2468
2242	One	Alt rock	4:36	DA	2122	2468
2233	So Cruel	Soft rock	5:49	NE	2122	2468
2361	Discothèque	Dance	5:19	NE	2123	2468
2372	Staring at the Sun	Pop	4:36	DA	2123	2468
2383	Mofo	Dance	5:49	NE	2123	2468
3181	Brianstorm	Punk rock	2:50	DA	3231	3579
3192	Teddy Picker	Punk rock	2:43	DA	3231	3579
3103	D is for Dangerous	Punk rock	2:16	NE	3231	3579
3241	My Propeller	Alt rock	3:27	NE	3232	3579
3252	Crying Lightning	Alt rock	3:43	DA	3232	3579
3263	Dangerous Animals	Alt rock	3:30	NE	3232	3579
3371	I Want It All	Rock	3:04	NE	3233	3579
3362	Arabella	Rock	3:27	NE	3233	3579
3353	Do I Wanna Know?	Rock	4:32	DA	3233	3579
4161	Bježi kišo s prozora	Novi val	3:16	DA	4341	4680
4172	Mojca, Mojca	Novi val	4:09	NE	4341	4680

4183	Dirlija	Novi val	4:00	DA	4341	4680
4291	Otrov	Pop rock	3:19	NE	4342	4680
4202	Tugo, nesrećo	Pop rock	3:14	NE	4342	4680
4213	Za sve ove godine	Pop rock	3:47	NE	4342	4680
4371	Zovu nas ulice	Pop rock	3:48	DA	4343	4680
4362	Ne daj na sebe	Pop rock	2:59	NE	4343	4680
4353	Ti znaš	Pop rock	5:06	NE	4343	4680
5171	Opća opasnost	Rock	3:33	DA	5451	5791
5182	Ne dirajte Županju	Rock	4:52	NE	5451	5791
5193	Jednom kad noć	Rock	4:07	DA	5451	5791
5201	Treba mi nešto jače od sna	Rock	4:02	DA	5452	5791
5212	Ti	Rock	4:19	NE	5452	5791
5223	Čudan ples	Rock	3:06	NE	5452	5791
5381	Mačka	Rock	4:23	DA	5453	5791
5372	Iskrena	Rock	6:05	NE	5453	5791
5363	U tvojim očima	Rock	4:07	NE	5453	5791

Tablica 4. Singlovi

SINGL				
ID pjesme	***ID albuma***	***Naziv singla***	***Žanr***	***Trajanje***
1162	1011	Creep	Soft rock	3:56
2131	2121	Where the Streets Have No Name	Rock	5:38
2142	2121	I Still Haven't Found What I'm Loking For	Gospel, rock	4:38
2153	2121	With or Without You	Soft rock	4:56
3181	3231	Brianstorm	Punk rock	2:50
3192	3231	Teddy Picker	Punk rock	2:43
4161	4341	Bježi kišo s prozora	Novi val	3:16
4183	4341	Dirlija	Novi val	4:00
5171	5451	Opća opasnost	Rock	3:33
5193	5451	Jednom kad noć	Rock	4:07
1291	1012	High and Dry	Indie rock	4:17
1363	1013	Karma Police	Alt rock	4:21
2242	2122	One	Alt rock	4:36
2372	2123	Staring at the Sun	Pop	4:36
3252	3232	Crying Lightning	Alt rock	3:43
3353	3233	Do I Wanna Know?	Rock	4:32
4371	4343	Zovu nas ulice	Pop rock	3:48
5201	5452	Treba mi nešto jače od sna	Rock	4:02
5381	5453	Mačka	Rock	4:23

Tablica 5. Članovi bendova

ČLAN BENDA				
ID benda	***ID člana***	***Ime člana***	***Prezime člana***	***Uloga u bendu***
1357	101	Thom	Yorke	Vokal, gitara
1357	102	Jonny	Greenwood	Solo gitara
1357	103	Colin	Greenwood	Bas gitara
1357	104	Ed	O'Brien	Ritam gitara
1357	105	Philip	Selway	Bubnjevi
2468	201	Bono	Vox	Vokal, gitara
2468	202	The	Edge	Gitara
2468	203	Adam	Clayton	Bas gitara
2468	204	Larry	Mullen, Jr.	Bubnjevi
3579	301	Alex	Turner	Vokal, gitara
3579	302	Jamie	Cook	Gitara
3579	303	Matt	Helders	Bubnjevi
3579	304	Nick	O'Malley	Bas gitara
4680	401	Dražen	Žerić	Vokal, klavijature
4680	402	Krešimir	Kaštelan	Bas gitara
4680	403	Igor	Matković	Klavijature
4680	404	Tomislav	Škrak	Gitara
4680	405	Adrian	Borić	Bubnjevi
5791	501	Pero	Galić	Vokal
5791	502	Slaven	Živanović	Gitara
5791	503	Igor	Kolić	Bas gitara
5791	504	Vlado	Soljačić	Gitara
5791	505	Josip	Kamenski	Bubnjevi
5791	506	Branimir	Jovanovac	Klavijature

2.1.3. Relacijski model

Konačno, u ovom je potpoglavlju predstavljen i relacijski model ogledne baze podataka (diskografske kuće). Model je, kao i onaj entiteta i veza, također napravljen prema metodologiji MIRIS [7].

BEND (ID, Naziv, Broj članova, Broj albuma, Broj pjesama, Broj singlova, Godina osnutka, Godina na sceni)

ALBUM (ID albuma, Naziv albuma, Broj pjesama, Žanr, Trajanje, Cijena, PDV, Ukupna cijena, ID)

PJESMA (ID pjesme, Naziv pjesme, Žanr, Trajanje, Je li singl, ID albuma, ID)

SINGL (ID pjesme, ID albuma, Naziv singla, Žanr, Trajanje)

ČLAN BENDA (ID, ID člana, Ime člana, Prezime člana, Uloga u bendu)

2.2. Varijable

Varijable u PL/SQL-u moraju biti deklarirane u DECLARE bloku ili u paketu[5] kao globalne varijable. Pri deklaraciji varijable, PL/SQL alocira memoriju za vrijednost varijable, a njeno je spremište identificirano imenom varijable. Ovako izgleda općenita sintaksa deklaracije varijable [4]:

```
Ime_varijable [CONSTANT] tip_podataka [NOT NULL] [:= | DEFAULT initial_value]
```

A ovo su primjeri nekoliko ispravnih deklaracija raznih varijabli:

```
cijena number(10, 2);
pi CONSTANT double precision := 3.1415;
adresa varchar2(100);
```

Ako varijabli u startu ne dodijelimo vrijednost, automatski joj je dodijeljena vrijednost NULL (po defaultu). Varijablu možemo inicijalizirati tijekom deklaracije, pomoću operatora pridruživanja ili ključne riječi DEFAULT [4];

```
brojac binary_integer := 0;
poruka varchar2(20) DEFAULT 'Lijep pozdrav';
```

[5] Poglavlje 2.13. – Paketi

PL/SQL dozvoljava ugniježđene blokove, odnosno svaki blok PL/SQL programa može sadržavati podblokove koji opet mogu imati svoje podblokove i tako dalje. Ako je varijabla deklarirana u unutarnjem bloku, njoj se ne može pristupiti iz vanjskog bloka. No, ako je deklarirana u vanjskom bloku, istoj varijabli može se pristupiti iz svih ugniježđenih (unutarnjih) blokova tog vanjskog bloka. Prema ovome, dijelimo varijable na lokalne (vrijede samo u unutarnjem bloku u kojem su deklarirane) i globalne (vrijede u svim unutarnjim blokovima vanjskog bloka u kojem su deklarirane) [4].

U primjeru koji slijedi demonstriran je doseg varijabli u PL/SQL-u, odnosno navedena razlika između lokalnih i globalnih varijabli. U početnom DECLARE bloku deklarirana je globalna varijabla 'broj_1' koja će vrijediti na razini cijelog programa, uključujući sve ugniježđene blokove izvršnog bloka. Ista varijabla, 'broj_1', ponovno je deklarirana unutar bloka ugniježđenog u izvršnom bloku te ona vrijedi samo u tom ugniježđenom bloku. Varijablama dodjeljujemo dvije različite vrijednosti te na kraju ispisom potvrđujemo da svaka ima svoj domet te stoga zadržavaju različite vrijednosti, unatoč tome što se praktički radi o istoj varijabli.

```
DECLARE

   -- globalna varijabla
   broj_1 number := 55;

BEGIN

   dbms_output.put_line('Vanjska varijabla broj_1: ' || broj_1);
   DECLARE
      -- lokalna varijabla
      broj_1 number := 155;
   BEGIN
      dbms_output.put_line('Unutarnja varijabla broj_1: ' || broj_1);
   END;

END;
/

Vanjska varijabla broj_1: 55
Unutarnja varijabla broj_1: 155

PL/SQL procedure successfully completed.
```

Primjer 3. Doseg varijabli u PL/SQL-u (globalne i lokalne varijable)

Pomoću SQL naredbe SELECT INTO, moguće je vrijednosti iz SQL tablica dodjeljivati PL/SQL varijablama. Pri tome je važno da se tipovi podataka podudaraju. Idući primjer pokazuje kako ispravno deklarirati varijable koje će biti istih tipova podataka kao retci u tablici PJESMA te kako deklariranim varijablama i pridružiti vrijednosti iz tablice PJESMA. 'Naziv_tablice.Naziv_stupca%type' izraz je koji upisujemo na mjesto gdje definiramo tip podataka varijable ukoliko želimo da varijabla poprimi tip podataka tog stupca te tablice.

```
DECLARE

p_ID_pjesme PJESMA.ID_PJESME%type := 2142;
p_Naziv_pjesme PJESMA.NAZIV_PJESME%type;
p_Zanr PJESMA.ZANR_PJESME%type;
p_Trajanje PJESMA.TRAJANJE%type;

BEGIN

   SELECT NAZIV_PJESME, ZANR_PJESME, TRAJANJE
INTO p_Naziv_pjesme, p_Zanr, p_Trajanje
   FROM PJESMA
   WHERE ID_PJESME = p_ID_pjesme;

   dbms_output.put_line
('Pjesma ' || p_Naziv_pjesme || ' zanra je ' || p_Zanr || ' i traje' ||
p_Trajanje || ' minuta.');

END;
/

Pjesma I Still Haven`t Found What I`m Looking For zanra je Gospel i traje
4:38 minuta.

PL/SQL procedure successfully completed.
```

Primjer 4. Dodjeljivanje vrijednosti iz SQL tablica PL/SQL varijablama

2.3. Tipovi podataka

Sve varijable, konstante i parametri u PL/SQL-u moraju biti nekog od važećih tipova podataka, koji definiraju način pohrane, ograničenja i opseg važećih vrijednosti [4]. Četiri su osnovna tipa podataka u PL/SQL-u:

- Skalari: mjerljive, jednostavne vrijednosti bez složenijih komponenti (npr. NUMBER, DATE, BOOLEAN)
- Veliki objekti (LOB – *Large Objects*, pokazivači na velike objekte koji se zasebno spremaju, kao što su tekst, slike, video isječci i zvuk)
- Složeni podaci: imaju unutarnje komponente kojima se može zasebno pristupiti, to su razne kolekcije i skupine zapisa
- Referencijalni tip podataka: pokazivači na druge objekte

Tipovi podataka koji spadaju u skalare su:

- Numerički
- Znakovni (*character*)
- Logički (*boolean*)
- Datum i vrijeme (*datetimes*)

U tablici 6 popisani su i opisani numerički tipovi podataka u PL/SQL-u.

Tablica 6. Numerički tipovi podataka u PL/SQL-u [4]

PLS_INTEGER	Vrijednosti od -2,147,483,648 do -2,147,483,648, 32 bita
BINARY_INTEGER	Vrijednosti od -2,147,483,648 do -2,147,483,648, 32 bita
BINARY_FLOAT	IEEE 754 standard jednostruke preciznosti
BINARY_DOUBLE	IEEE 754 standard jednostruke preciznosti
NUMBER (prec, scale)	Vrijednosti od 1E-130 do (ne uključujući) 1.0E126, može biti i 0
DEC (prec, scale)	Vezan uz ANSI[6], maksimalna preciznost 38 decimala
DECIMAL (prec, scale)	Vezan uz IBM, maksimalna preciznost 38 decimala
NUMERIC (prec, scale)	IEEE 754 standard, maksimalna preciznost 38 decimala
DOUBLE PRECISION	Vezan uz ANSI, IEEE 754 standard, max. preciznost 38 dec.
FLOAT	Vezan uz ANSI I IBM, IEEE 754 standard, maxsimalna preciznost 38 decimala
INT	INTEGER tip vezan uz ANSI, max. preciznost 38 dec.
INTEGER	INTEGER tip vezan uz ANSI i IBM, max. preciznost 38 dec.
SMALLINT	INTEGER tip vezan uz ANSI i IBM, max. preciznost 38 dec.
REAL	IEEE 754 standard, maksimalna preciznost 18 decimala

[6] American National Standards Institute

U idućem primjeru predstavljena je ispravna deklaracija varijabli za neke od numeričkih tipova podataka u PL/SQL-u. Također je prikazan i ispis tih varijabli.

```
DECLARE

      broj_1 INT := 100;
      broj_2 FLOAT := 5.6;
      broj_3 BINARY_DOUBLE := 560;
      broj_4 PLS_INTEGER := 5;

BEGIN

      dbms_output.put_line('Broj 1: ' || broj_1);
      dbms_output.put_line('Broj 2: ' || broj_2);
      dbms_output.put_line('Broj 3: ' || broj_3);
      dbms_output.put_line('Broj 4: ' || broj_4);

END;
/

Broj 1: 100
Broj 2: 5,6
Broj 3: 5,6E+002
Broj 4: 5

PL/SQL procedure successfully completed.
```

Primjer 5. Ispravna deklaracija i ispis za neke od numeričkih tipova podataka

U tablici 7 popisani su i opisani znakovni tipovi podataka u PL/SQL-u.

CHAR	Fiksne duljine, maksimalne veličine 32,767 bajtova
VARCHAR2	Varijabilne duljine, maksimalne veličine 32,767 bajtova
NCHAR	UTF-8 / UTF16BE, Fiksne duljine, maksimalne veličine 32,767 bajtova
NVARCHAR2	UTF-8 / UTF16BE, Varijabilne duljine, maksimalne veličine 32,767 bajtova
LONG	Varijabilne duljine, maksimalne veličine 32,760 bajtova
ROWID	Identifikator retka u tablici
UROWID	Univerzalni identifikator retka u tablici

Tablica 7. Znakovni tipovi podataka u PL/SQL-u [4]

U primjeru je demonstrirana ispravna deklaracija i ispis varijabli za neke od znakovnih tipova podataka u PL/SQL-u.

```
DECLARE

      tekst_1 varchar2(10) := 'Poruka!';
      tekst_2 char := 'D';
      tekst_3 long(15) := 'Treca poruka!';

BEGIN

      dbms_output.put_line('Tekst 1: ' || tekst_1);
      dbms_output.put_line('Tekst 2: ' || tekst_2);
      dbms_output.put_line('Tekst 3: ' || tekst_3);

END;
/

Tekst 1: Poruka!
Tekst 2: D
Tekst 3: Treca poruka!

PL/SQL procedure successfully completed.
```

Primjer 6. Ispravna deklaracija i ispis za neke od znakovnih tipova podataka

Logički tip podataka BOOLEAN koristimo za spremanje vrijednosti koje se koriste u logičkim operacijama, a one mogu biti TRUE, FALSE ili NULL. No, budući da u SQL-u ne postoji tip podataka ekvivalentan ovome, BOOLEAN ne možemo koristiti u SQL naredbama, ugrađenim SQL funkcijama niti u PL/SQL funkcijama izvedenih iz SQL naredbi [4].

U tablici 8 predstavljeni su veliki objekti (LOB – Large Objects), poseban tip podataka u PL/SQL-u koji zapravo predstavlja pokazivače na velike objekte koji se zasebno spremaju, kao što su tekst, slike, video-isječci i zvuk.

Tablica 8. Veliki objekti (LOB - Large Objects) [4]

BFILE	Koristi se za pohranu velikih binarnih objekata u datotekama operacijskog sustava izvan baze podataka	Veličina ovisi o sustavu, ne može prijeći 4 GB
BLOB	Koristi se za pohranu velikih binarnih objekata unutar baze podataka	Veličine 8 do 128 terabajta
CLOB	Koristi se za pohranu velikih blokova znakovnih podataka unutar baze	Veličine 8 do 128 TB
NCLOB	Koristi se za pohranu velikih blokova NCHAR podataka unutar baze	Veličine 8 do 128 TB

2.4. Konstante

Konstante se u PL/SQL-u deklariraju uz ključnu riječ CONSTANT. Zahtijevaju početnu vrijednost odmah pri deklaraciji i tu je vrijednost kasnije nemoguće mijenjati [4].

Primjer:

```
PI CONSTANT NUMBER := 3.141592654;
```

Kada se radi o jednom znaku ili nizu znakova koji ne sadrži varijable, njihove nazive ili bilo kakav dio koda, već želimo te znakove doslovno ispisati kao tekst, taj ćemo tekst staviti u navodnike. Postoje razni znakovi koji se u PL/SQL-u interpretiraju doslovno, raznih tipova podataka.

U idućoj tablici predstavljeni su primjeri znakova koji se u PL/SQL-u interpretiraju doslovno, podijeljeni po tipovima podataka [4]:

Tablica 9. Znakovi koji se interpretiraju doslovno u PL/SQL-u

Numerički	050 78 -14 0 +32767 6.6667 0.0 -12.0 3.14159 +7800.00 6E5 1.0E-8 3.14159e0 -1E38 -9.5e-3
Character (jedan znak)	'B' '&' '25' ' ' 'x' '}'
String (više znakova)	'Hello, world!' 'PL/SQL' '19-NOV-12'
BOOLEAN (logički)	TRUE FALSE i NULL
Date and time (datum i vrijeme)	DATE '1978-12-25'; TIMESTAMP '2012-10-29 12:01:01';

2.5. Operatori

Ovo su operatori koji se koriste u PL/SQL-u [4]:

- Aritmetički operatori:
  - + (zbrajanje; a+b)
  - - (oduzimanje; a-b)
  - * (množenje; a*b)
  - / (dijeljenje; a/b)
  - ** (potenciranje; a**b)

- Relacijski operatori:
  - = (provjera jesu li dvije vrijednosti jednake, ako jesu uvjet je TRUE)
  - != / <> / ~ = (ako dvije vrijednosti nisu jednake uvjet je TRUE)
  - > (usporedba vrijednosti, veće od)
  - < (usporedba vrijednosti, manje od)
  - <= (manje ili jednako)
  - >= (veće ili jednako)

- Operatori usporedbe:
  - LIKE: služi za usporedbu bilo kakvog znakovnog niza sa nekom šablonom, primjer; 'Saba Ali' LIKE 'S% A_i' vratiti će TRUE
  - BETWEEN: provjerava je li vrijednost dio nekog intervala, npr. x BETWEEN 5 and 10 biti će TRUE samo ako je x >=5 i x<=10
  - IN: je li vrijednost dio zadanog niza, odnosno odgovara li bilo kojoj od nabrojanih vrijednost, npr. Ako je x = 'g', x IN ('a', 'b', 'c') biti će FALSE, a x IN ('f', 'g', 'h') biti će TRUE
  - IS NULL: biti će TRUE samo ako je vrijednost NULL

- Logički operatori:
  - and (povezuje uvjete, vraća TRUE ako su oba uvjeta istinita)
  - or (vraća TRUE ako je bar jedan od uvjeta istinit)
  - not (negacija, npr. ako je (A and B) TRUE, NOT (A and B) biti će FALSE)

U tablici su operatori poredani po prioritetima, odnosno u bilo kakvom izrazu s više operatora prvo će se izvršiti oni na vrhu tablice, a na kraju oni na dnu. Naravno, u takvim složenijim izrazima, prije svega navedenog najveći prioritet imaju zagrade. Što se tiče operatora za uspoređivanje, njihov je prioritet kako slijedi, od višeg prema manjem prioritetu:

=, <, >, <=, >=, <>, !=, ~=, ^=, IS NULL, LIKE, BETWEEN, IN [4].

Tablica 10. Prioriteti operatora u PL/SQL-u [4]

****	potenciranje
*, /	množenje, dijeljenje
+, -, ||	zbrajanje, oduzimanje, konkatenacija
Operatori usporedbe i relacijski operatori	
NOT	logička negacija
AND	Konjunkcija
OR	disjunkcija

2.6. Uvjeti

Fraza IF-THEN-ELSE omogućuje ocjenu varijable po nekom kriteriju te poduzimanje određene akcije na temelju zadanog kriterija. Ovo je opća sintaksa [1]:

```
IF  if_uvjet  THEN
      if_blok  |  NULL;
[  ELSIF  elsif_uvjet  THEN
      elsif_blok  |  NULL;  ]…
[  ELSE  else_uvjet  THEN
      else_blok  |  NULL;
```

Pod `if_uvjet` mora biti uvjet koji poprima vrijednosti TRUE ili FALSE. Pod `if_blok` mora biti niz ispravnih PL/SQL naredbi koje će se izvršiti u slučaju istinitosti uvjeta. Koristimo NULL ukoliko ne želimo da se išta izvrši u ovom koraku [1].

ELSIF dio je neobavezan, a može se i ponavljati koliko god puta želimo. Pod `elsif_uvjet` također se misli na uvjet koji može biti TRUE ili FALSE, a provjeravati će se tek ukoliko `if_uvjet` nije TRUE. Isto tako, pod `elsif_blok` radi se o bloku ispravnih PL/SQL naredbi koje će se izvršiti ukoliko je ovaj uvjet istinit. Ako ne želimo da se išta izvrši u ovom koraku, koristimo naredbu NULL [1].

ELSE dio također je neobavezan, ali često korišten pošto se odnosi na PL/SQL naredbe koje će se izvršiti ako niti jedan od zadanih uvjeta nije istinit.

```
DECLARE

      Iznos FLOAT := 523.60;
      Status varchar2(30);

BEGIN
      IF Iznos < 0 THEN
            Status := 'Iznos je negativan!';
      ELSIF Iznos = 0 THEN
            Status := 'Iznos je jednak nuli!';
      ELSE
            Status := 'Iznos je pozitivan!';
      END IF;
      dbms_output.put_line(Status);
END;
/

Iznos je pozitivan!

PL/SQL procedure successfully completed.
```

Primjer 7. IF-THEN-ELSE struktura

2.7. Kursori

„Kursor baze podataka je privatno SQL područje u kojem se čuvaju informacije za procesiranje određene SQL naredbe. Oracle PL/SQL jezik koristi implicitne i eksplicitne kursore [2].“

Kursori se koriste za prebacivanje informacija iz baze podataka u PL/SQL program. Kursor sadrži SQL naredbu SELECT. Postoje dvije vrste kursora; eksplicitni i implicitni [1].

2.7.1. Eksplicitni kursori

Eksplicitni kursor zahtijeva definiranu izjavu. Mora ga se programirati tako da se otvori, prigrabi podatke i zatvori se [1]. Ovako izgleda opća sintaksa eksplicitnog kursora:

DECLARE CURSOR ime_kursora IS

Kursori se koriste kroz tri koraka [1]:

1. **Otvaranje kursora:**

```
OPEN ime_kursora;
```

2. ***Grabljenje* podataka jednog po jednog retka tablice:**

```
FETCH ime_kursora INTO ime_varijable_1, ime_varijable_2, …;
```

Za svaki novi redak koristimo novu naredbu FETCH. Kursor ne zatvaramo sve dok ne završimo sa svim recima. Ovdje je iznimno praktično koristiti petlju[7].

3. **Zatvaranje kursora:**

```
CLOSE ime_kursora;
```

Primjer 8 prikazuje korištenje eksplicitnog kursora (bez petlje) za dohvaćanje podataka iz prvih nekoliko redaka tablice BEND, konkretno broja pjesama. Na temelju dohvaćenih podataka računa se ukupan broj pjesama svih bendova čiji su podaci dohvaćeni.

```
DECLARE

      V_ID INT;
      V_BR_PJESAMA INT;
      V_UKUPAN_BR_PJESAMA INT := 0;
      CURSOR BENDOVI IS
      SELECT ID, BR_PJESAMA FROM BEND;

BEGIN

      -- otvaranje kursora
      OPEN BENDOVI;
      -- grabljenje prvog retka
      FETCH BENDOVI INTO V_ID, V_BR_PJESAMA;
            V_UKUPAN_BR_PJESAMA := V_UKUPAN_BR_PJESAMA + V_BR_PJESAMA;
     -- grabljenje drugog retka
      FETCH BENDOVI INTO V_ID, V_BR_PJESAMA;
            V_UKUPAN_BR_PJESAMA := V_UKUPAN_BR_PJESAMA + V_BR_PJESAMA;
      -- grabljenje treceg retka
      FETCH BENDOVI INTO V_ID, V_BR_PJESAMA;
            V_UKUPAN_BR_PJESAMA := V_UKUPAN_BR_PJESAMA + V_BR_PJESAMA;
      -- zatvaranje kursora
      CLOSE BENDOVI;
      -- ispis ukupnog broja pjesama
      dbms_output.put_line('Ukupan broj pjesama: ' || V_UKUPAN_BR_PJESAMA);

END;
/

Ukupan broj pjesama: 314
PL/SQL procedure successfully completed.
```

Primjer 8. Eksplicitni kursor

[7]Petlje – poglavlje 2.8.

2.7.2. Implicitni kursori

Implicitni kursor je bolje staviti u tijelo, nego u blok deklaracije PL/SQL programa. Implicitni kursor SELECT mora vratiti vrijednost jednog retka. Ukoliko upit vraća više od jednog retka, potrebno je promijeniti upit ili koristiti eksplicitni kursor. Ovako izgleda opća sintaksa implicitnog kursora [1]:

```
SELECT stupac_1, stupac_2 … INTO varijabla_1, varijabla_2, … FROM …
```

Osim naredbe SELECT, u implicitnom se kursoru mogu pojavljivati i naredbe INSERT, UPDATE i DELETE.

Primjer 9 prikazuje računanje ukupnog broja pjesama svih bendova u tablici BEND korištenjem implicitnog kursora. Najprije u bloku DECLARE deklariramo varijablu 'v_ukupan_br_pjesama' numeričkog cjelobrojnog tipa INT u koju ćemo kursorom 'pokupiti' podatke iz tablice. Zatim u izvršnom bloku podatke o sumi po stupcu 'br_pjesama' u tablici BEND spremamo u varijablu 'v_ukupan_br_pjesama' kako smo i pokazali u općoj sintaksi implicitnog kursora. U konačnici vršimo i ispis varijable 'v_ukupan_br_pjesama' te uistinu dobivamo ukupan broj svih pjesama u tablici BEND, odnosno sumu po stupcu 'br_pjesama' u ovoj tablici.

```
DECLARE

      V_UKUPAN_BR_PJESAMA INT;

BEGIN

      -- implicitni kursor
      SELECT SUM(BR_PJESAMA)
      INTO V_UKUPAN_BR_PJESAMA
      FROM BEND;

      -- ispis ukupnog broja pjesama
      dbms_output.put_line('Ukupan broj pjesama: ' || V_UKUPAN_BR_PJESAMA);

END;
/

Ukupan broj pjesama: 559

PL/SQL procedure successfully completed.
```

Primjer 9. Implicitni kursor

2.7.3. Atributi kursora

Postoje četiri atributa koja nam govore o stanju kursora. Svaki se poziva konkatenacijom na ime kursora. To su, redom [1]:

- ime_kursora%FOUND: vraća TRUE kada je redak uspješno prigrabljen posljednjom naredbom FETCH
- ime_kursora%NOTFOUND: suprotan atributu FOUND, vraća TRUE onda kada redak nije prigrabljen posljednjom naredbom FETCH
- ime_kursora%ISOPEN: vraća TRUE ako je kursor bio otvoren kada je izvedena naredba FETCH, vraća FALSE prije otvaranja i zatvaranja kursora
- ime_kursora%ROWCOUNT: vraća broj redaka koji su do sada uspješno prigrabljeni, odnosno koliko je redaka trenutno u kursoru

U primjeru 10 testirani su atributi kursora. U istom primjeru demonstrirane su i dvije dodatne osobine [1]:

- Naredba GOTO, koja nam omogućuje da se izravno prebacimo na drugo mjesto u PL/SQL kodu, definirano oznakom.
- Oznaka (sintaksa: <<oznaka>>). Izjava GOTO upućuje nas na oznaku. Ispod oznake mora se nalaziti minimalno jedna naredba.

Kursori su moćan programski alat koji nam otvara nebrojeno mogućnosti rada s bazom podataka direktno iz PL/SQL programa, što olakšava i skraćuje sam rad s bazom za prosječnog korisnika. Međutim, da bi iskoristili njihov puni potencijal, potrebno ih je koristiti u kombinaciji s petljama[8]. One su tema idućeg poglavlja.

```
DECLARE
      V_ID INT;
      V_BR_PJESAMA INT;
      V_UKUPAN_BR_PJESAMA INT := 0;
      CURSOR BENDOVI IS
      SELECT ID, BR_PJESAMA FROM BEND;
BEGIN
      -- otvaranje kursora
      OPEN BENDOVI;
      -- grabljenje prvog retka
      FETCH BENDOVI INTO V_ID, V_BR_PJESAMA;
      IF BENDOVI%FOUND THEN
            V_UKUPAN_BR_PJESAMA := V_UKUPAN_BR_PJESAMA + V_BR_PJESAMA;
      ELSE
            GOTO CLOSE_THE_CURSOR;
      END IF;
      -- grabljenje drugog retka
      FETCH BENDOVI INTO V_ID, V_BR_PJESAMA;
      IF BENDOVI%ROWCOUNT = 2 THEN
            V_UKUPAN_BR_PJESAMA := V_UKUPAN_BR_PJESAMA + V_BR_PJESAMA;
      ELSE
            GOTO CLOSE_THE_CURSOR;
      END IF;
      <<CLOSE_THE_CURSOR>>
      -- zatvaranje kursora
      CLOSE BENDOVI;
      -- ispis ukupnog broja pjesama
      dbms_output.put_line('Ukupan broj pjesama: ' || V_UKUPAN_BR_PJESAMA);
END;
/

Ukupan broj pjesama: 249

PL/SQL procedure successfully completed.
```

Primjer 10. Testiranje atributa kursora

[8] Poglavlje 2.8. – Petlje

2.8. Petlje

2.8.1. FOR petlja kursora

Ova petlja jednostavno rukuje kursorima i ponavlja kod za svaki redak čije vrijednosti preuzima kursor. Bez problema rukuje naredbama kursora OPEN, FETCH i CLOSE. Ovako izgleda opća sintaksa [1]:

```
FOR ime_retka IN ime_kursora LOOP
blok_petlje
END LOOP;
```

U Primjeru 11 demonstrirana je FOR petlja kursora. Najprije u bloku deklaracija deklariramo kursor 'pjesme' u koji će se spremiti svi podatci iz stupaca 'ID_pjesme' i 'naziv_pjesme' iz tablice PJESMA. Zatim pomoću FOR petlje u dva stupca ispisujemo te podatke o ID-u i nazivu svih pjesama iz tablice PJESMA koje se sada nalaze u kursoru 'pjesme'. U ovom primjeru nismo morali koristiti naredbe OPEN, FETCH i CLOSE te se ovdje radi o jednostavnijoj varijanti eksplicitnoga kursora, a koju možemo koristiti kada god imamo FOR petlju. Dovoljno je da unutar definicije FOR petlje deklariramo novu varijablu (u ovom slučaju 'redak') koja će nam označavati jedan redak koji kursor dohvaća. Zatim, ukoliko imamo više stupaca u kursoru, svaki pojedini stupac označujemo s 'redak.ime_stupca'.

```
DECLARE
      CURSOR pjesme IS
      SELECT ID_PJESME, NAZIV_PJESME
      FROM PJESMA;
BEGIN
      FOR redak IN pjesme
      LOOP
      DBMS_OUTPUT.PUT_LINE
      (redak.ID_PJESME || ' ' ||
      redak.NAZIV_PJESME);
      END LOOP;
END;
/

1151 You
1162 Creep
1173 How Do You?
1291 High and Dry
...
```

Primjer 11. FOR petlja kursora – jednostavniji primjer

U nastavku imamo i nešto složeniji primjer. U ovome primjeru kombinacijom kursora i FOR petlje, odnosno korištenjem FOR petlje kursora, umećemo novi redak u tablicu SINGL za sve pjesme iz tablice PJESMA koje u stupcu JE_LI_SINGL imaju vrijednost 'DA', koristeći pritom vrijednosti iz tablice PJESMA, budući da sve stupce u tablici SINGL imamo i u tablici PJESMA. U bloku deklaracija deklariramo implicitni kursor 'pjesme' u koji spremamo stupce 'id_pjesme', 'naziv_pjesme', 'je_li_singl', 'zanr_pjesme', 'trajanje' i 'id_albuma' iz tablice PJESMA. Zatim u izvršnom bloku provlačimo kursor 'pjesme' kroz for petlju te najprije postavljamo uvjet. Ispitujemo za svaki pojedini redak u kursoru vrijednost stupca 'je_li_singl'. Ukoliko je ta vrijednost jednaka 'DA' svi podatci iz kursora umeću se kao novi redak u tablicu SINGL, pri čemu 'naziv_pjesme' postaje 'naziv_singla', 'zanr_pjesme' je 'zanr_singla', a 'trajanje' je 'trajanje_singla'.

```
DECLARE

      CURSOR pjesme IS
      SELECT ID_PJESME, NAZIV_PJESME, JE_LI_SINGL, ZANR_PJESME,
      TRAJANJE, ID_ALBUMA
      FROM PJESMA;

BEGIN

      FOR redak IN pjesme
      LOOP

            IF redak.JE_LI_SINGL = 'DA' THEN
                  INSERT INTO SINGL
                  (ID_PJESME, ID_ALBUMA, NAZIV_SINGLA,
                  ZANR_SINGLA, TRAJANJE_SINGLA)
                  VALUES
                  (redak.ID_PJESME, redak.ID_ALBUMA,
                  redak.NAZIV_PJESME, redak.ZANR_PJESME,
                  redak.TRAJANJE);
            END IF;

      END LOOP;
END;
/

PL/SQL procedure successfully completed.
```

Primjer 12. FOR petlja kursora – složeniji primjer (nad bazom podataka)

2.8.2. Jednostavna petlja

Ovako izgleda opća sintaksa jednostavne petlje [1]:

```
LOOP
blok_petlje
END LOOP;
```

Tehnički gledano, jednostavna petlja nema kraja ali možemo ju zaustaviti naredbama EXIT ili EXIT WHEN. Naredba EXIT ide nakon završenog testiranja nekog uvjeta, a EXIT WHEN je uvjet koji dodajemo na kraj bloka petlje te ga petlja testira svaki put kada naiđe na tu izjavu [1].

U Primjeru 13, u tablici ALBUM, broj pjesama na albumu ulazi u petlju sve dok ukupan zbroj ne dosegne 100. Također, ukoliko se petlja ponovi 9 puta, odnosno prođe kroz 9 albuma prije nego dosegne broj od 100 pjesama, biti će prekinuta.

```
DECLARE
      V_BROJ_PJESAMA INT;
      V_BROJ_PETLJI INT := 0;
      V_ZBROJ_PJESAMA INT := 0;
            CURSOR albumi IS
            SELECT BR_PJESAMA FROM ALBUM;
BEGIN
      FOR redak IN albumi
      LOOP
      V_BROJ_PETLJI := V_BROJ_PETLJI + 1;
      V_ZBROJ_PJESAMA := V_ZBROJ_PJESAMA + redak.BR_PJESAMA;
      IF V_ZBROJ_PJESAMA > 100 THEN
            -- prekid petlje ako smo došli do 100 pjesama
            -- ispis zbroja pjesama
            dbms_output.put_line ('Zbroj pjesama: ' ||
V_ZBROJ_PJESAMA);
            EXIT;
      ELSIF V_BROJ_PETLJI > 9 THEN
            -- prekid ukoliko se petlja odvrtila 9 puta
            -- ispis zbroja pjesama
            dbms_output.put_line ('Zbroj pjesama: ' ||
V_ZBROJ_PJESAMA);
            EXIT;
      END IF;
      END LOOP;
END;
/

Zbroj pjesama: 103

PL/SQL procedure successfully completed.
```

Primjer 13. Jednostavna petlja

2.8.3. While petlja

Ova petlja vrti se sve dok određeni zadani uvjet ne postane istinit. Ovako izgleda njena opća sintaksa [1]:

```
WHILE uvjet LOOP
blok_petlje
END LOOP;
```

Primjer 14 predstavlja problem iz primjera 14 riješen korištenjem WHILE petlje.

```
DECLARE

      V_BROJ_PJESAMA INT;
      V_BROJ_PETLJI INT := 0;
      V_ZBROJ_PJESAMA INT := 0;
            CURSOR albumi IS
            SELECT BR_PJESAMA FROM ALBUM;
BEGIN

      OPEN albumi;
      WHILE (V_ZBROJ_PJESAMA <= 100 AND V_BROJ_PETLJI <= 9) LOOP
      V_BROJ_PETLJI := V_BROJ_PETLJI + 1;
            FETCH albumi INTO V_BROJ_PJESAMA;
            V_ZBROJ_PJESAMA := V_ZBROJ_PJESAMA + V_BROJ_PJESAMA;
      END LOOP;

      CLOSE albumi;
      dbms_output.put_line('Zbroj pjesama je: ' || V_ZBROJ_PJESAMA);
END;
/

Zbroj pjesama je: 103

PL/SQL procedure successfully completed.
```

Primjer 14. While petlja

2.8.4. FOR petlja

Ovu petlju koristimo kada znamo točan broj ponavljanja petlje. Potrebno je posebno deklarirati varijablu (izvan bloka DECLARE, unutar same petlje) pomoću koja će nositi vrijednost duljine petlje. Ovako izgleda opća sintaksa [1]:

```
FOR countvar IN [REVERSE] start_num  .  .  end_num LOOP
blok_petlje
END LOOP;
```

Pri ovome countvar predstavlja varijablu duljine petlje, a start_num i end_num numeričke su ili konstantne varijable koje predstavljaju početnu i krajnju granicu petlje, odnosno raspon vrijednosti koje će poprimati varijabla countvar u svakom idućem ponavljanju petlje (u primjeru v_broj_petlji in 1 . . 9 => 9 ponavljanja petlje).

Ponovno se vraćamo na isti problem te ćemo ga u primjeru 15 riješiti korištenjem FOR petlje tako da unaprijed zadamo broj ponavljanja petlje koji će biti 9. Pomoću IF uvjeta prekinuti ćemo petlju ranije ukoliko se dođe do zbroja od 100 pjesama prije 9. ponavljanja petlje.

```
DECLARE
      V_BROJ_PJESAMA INT;
      V_ZBROJ_PJESAMA INT := 0;
            CURSOR albumi IS
            SELECT BR_PJESAMA FROM ALBUM;
BEGIN

      OPEN albumi;
      FOR V_BROJ_PETLJI IN 1 .. 9 LOOP
            FETCH albumi INTO V_BROJ_PJESAMA;
            V_ZBROJ_PJESAMA := V_ZBROJ_PJESAMA + V_BROJ_PJESAMA;
            IF V_ZBROJ_PJESAMA > 100 THEN
                  -- raniji prekid petlje ako se dođe do 100 pjesama
                  EXIT;
            END IF;
      END LOOP;
      CLOSE albumi;
            dbms_output.put_line('Zbroj pjesama je: ' ||
            V_ZBROJ_PJESAMA);
END;
/

Zbroj pjesama je: 103

PL/SQL procedure successfully completed.
```

Primjer 15. FOR petlja

2.9. Stringovi

Stringovi u PL/SQL-u nizovi su znakova s neobaveznom specifikacijom duljine. Ti nizovi mogu sadržavati brojeve, slova, prazne znakove (razmake), posebne znakove ili bilo kakvu kombinaciju svega navedenog. Postoje tri vrste stringova u PL/SQL-u [4]:

- Stringovi fiksne duljine: programer određuje duljinu stringa pri njegovoj deklaraciji te se on zdesna puni razmacima sve do zadane duljine
- Stringovi varijabilne duljine: programer određuje maksimalnu duljinu stringu odnosno najveći broj znakova koji se može unijeti, no duljina stringa varira ovisno o unesenom broju znakova te se ovakav string ne puni razmacima, najveća moguća duljina koja se može zadati za ovakav string je 32,767
- Znakovi velikih objekata (CLOBs – *Character Large Objects*): stringovi varijabilne duljine, moguće veličine sve do 128 terabajta

Stringovi u PL/SQL-u mogu biti varijable ili literali. Stringovi literali pišu se unutar jednostrukih navodnika [4]. Pri tome se misli na tekst koji će se doslovno ispisati na ekranu bez da se posebno sprema u varijablu. Na primjer:

```
'Ovo je jedan takav string.'
```

Ukoliko želimo imati jednostruki navodnik unutar ovakvog stringa, da bi PL/SQL taj navodnik interpretirao doslovno moramo dva jednostruka navodnika napisati jedan za drugim. Na primjer:

```
'Ovo je jedan "takav" string.'
```

U Oracle bazi postoji nekoliko tipova podataka za stringove, kao npr. CHAR, NCHAR, VARCHAR2, NVARCHAR2, CLOB i NCLOB. Prefiks 'N' označuje 'national character set' tipove podataka, koji spremaju podatke za Unicode znakove.

Pri deklaraciji stringa varijabilne duljine, potrebno je odrediti maksimalnu moguću duljinu tog stringa. U primjeru 16[9] predstavljene su deklaracije nekoliko različitih tipova stringova. Varijable 'ime' i 'mjesto' tipa su varchar2, varijabilne duljine, a maksimalni broj znakova koje u njih možemo spremiti je 20, odnosno 30. Varijabla 'pozdrav' tipa je clob, što znači da se radi o ogromnom stringu varijabilne duljine, a čija veličina može biti čak do 128 terajabajta. Varijabla 'znak' tipa je char, idealan tip za spremanje samo jednog znaka što nam u ovom slučaju i treba. Na kraju sve četiri varijable ispisujemo tako da ih prosljeđujemo kao parametar naredbi 'dbms_output.put_line'. Pri tome znakom '||' odvajamo vrijednost varijable od teksta koji želimo da se doslovno ispiše prije ili poslije varijable.

[9]Na idućoj stranici

```
DECLARE

ime varchar2(20);
mjesto varchar2(30);
pozdrav clob;
znak char(1);

BEGIN

ime := 'Ivan Ivic';
mjesto := 'Rijeka';
pozdrav := 'Pozdrav! Ja sam Ivan Ivic iz Rijeke.';
znak := 'y';

   IF znak = 'y' THEN

      dbms_output.put_line('Ime: ' || ime);
      dbms_output.put_line('Mjesto: ' || mjesto);
      dbms_output.put_line('Znak: ' || znak);
      dbms_output.put_line('Pozdrav: ' || pozdrav);

   END IF;

END;
/

Ime: Ivan Ivic
Mjesto: Rijeka
Znak: y
Pozdrav: Pozdrav! Ja sam Ivan Ivic iz Rijeke.

PL/SQL procedure successfully completed.
```

Primjer 16. Stringovi

U sljedećem primjeru prikazane su (uz objašnjenja što rade u komentarima) neke od najčešće korištenih gotovih PL/SQL funkcija nad stringovima:

```
DECLARE

      poruka varchar2(20) := 'Ovo je neki string';

BEGIN

      -- pretvara sva slova u velika
      dbms_output.put_line(UPPER(poruka));

      -- pretvara sva slova u mala
      dbms_output.put_line(LOWER(poruka));

      -- pretvara sva pocetna slova u velika
      dbms_output.put_line(INITCAP(poruka));

      -- vraca prvi znak u stringu
      dbms_output.put_line (SUBSTR (poruka, 1, 1));

      -- vraca posljednji znak u stringu
      dbms_output.put_line (SUBSTR (poruka, -1, 1));

      -- vraca 4 znaka u stringu, pocevsi od 8. pozicije
      dbms_output.put_line (SUBSTR (poruka, 8, 4));

      -- ispisuje na kojem se mjestu unutar stringa nalazi prvi znak e
      dbms_output.put_line (INSTR (poruka, 'e'));

END;
/

OVO JE NEKI STRING
ovo je neki string
Ovo Je Neki String
O
g
neki
6

PL/SQL procedure successfully completed.
```

Primjer 17. Funkcije nad stringovima

2.10. Polja

Za polja u PL/SQL-u koristimo tip podataka VARRAY, u koji je moguće spremiti sekvencijalni niz podataka istoga tipa. Ovakvo polje fiksne je veličine, a svaki element polja ima svoj jedinstveni indeks i maksimalnu veličinu koja se može dinamički mijenjati [4].

Tip podataka VARRAY stvara se pomoću naredbe CREATE TYPE. Potrebno je specificirati maksimalnu veličinu polja i kojeg će tipa biti podaci u njemu. Ovako izgleda opća sintaksa [1]:

```
CREATE OR REPLACE TYPE ime_polja IS VARRAY(n) of <tip_podataka>;
```

Pri čemu je *n* maksimalan broj elemenata u polju. Konkretnije, unutar PL/SQL bloka ovako stvaramo polja [4]:

```
TYPE ime_polja IS VARRAY(n) of <tip_podataka>
TYPE polje_tekst IS VARRAY(5) OF VARCHAR2(10);
TYPE polje_brojevi IS VARRAY(5) OF INT;
```

Primjer 18 jednostavan je primjer u kojem ćemo napraviti 2 polja, jedno za studente i jedno za njihove bodove, te ispisati podatke, odnosno bodove za svakog studenta.

```
DECLARE

   type STUDENTI_POLJE IS VARRAY(4) OF VARCHAR2(10);
   type BODOVI_POLJE IS VARRAY(4) OF INT;
STUDENTI STUDENTI_POLJE;
BODOVI BODOVI_POLJE;
BR_STUDENATA INT;

BEGIN

STUDENTI := STUDENTI_POLJE('Ivan', 'Marko', 'Luka', 'Tin');
BODOVI:= BODOVI_POLJE(84, 53, 72, 61);
BR_STUDENATA := STUDENTI.count;
   dbms_output.put_line('Ukupno '|| BR_STUDENATA || ' studenta');
   FOR i in 1 .. BR_STUDENATA LOOP
      dbms_output.put_line('Student: ' || STUDENTI(i) || ',
                           Bodovi : ' || BODOVI(i));
   END LOOP;

END;
/

Ukupno 4 studenta
Student: Ivan, Bodovi : 84
Student: Marko, Bodovi : 53
Student: Luka, Bodovi : 72
Student: Tin, Bodovi : 61

PL/SQL procedure successfully completed.
```

Primjer 18. Polja – jednostavan primjer

Primjer 19 nešto je složeniji primjer u kojem ćemo polje povezati sa našom bazom podataka. Pomoću kursora stvorit ćemo jedno polje na temelju tablice ČLAN BENDA te u njega pospremiti imena svih glazbenika (iz stupca 'ime_člana') u tablici i ispisati ih.

```
DECLARE

   CURSOR clanovi IS
   SELECT IME FROM CLAN_BENDA;
TYPE V_POLJE is varray(30) of CLAN_BENDA.IME%type;
   /* ovim retkom stvoreno je polje naziva V_POLJE max. velicine 30,
   * istog tipa podataka kao i stupac IME u tablici CLAN_BENDA
   */
IME_LIST V_POLJE := V_POLJE();
BROJAC INT :=0;

BEGIN

   FOR redak IN clanovi LOOP
BROJAC := BROJAC + 1;
IME_LIST.extend;
IME_LIST(BROJAC)  := redak.IME;
      dbms_output.put_line('CLAN('|| BROJAC ||'):'||IME_LIST(BROJAC));
   END LOOP;

END;
/

CLAN(1): Thom
CLAN(2): Jonny
CLAN(3): Colin
CLAN(4): Ed
CLAN(5): Philip
CLAN(6): Bono
...

PL/SQL procedure successfully completed.
```

Primjer 19. Polja – složeniji primjer (povezivanje s tablicom iz baze)

2.11. Procedure

PL/SQL potprogrami zasebni su imenovani PL/SQL blokovi koji se mogu pozvati iz glavnog programa tako da im se proslijedi skup parametara. Oni se spremaju na server kao objekti baze podataka te ih se može pozvati dok god se nalaze na serveru. Postoje dvije vrste takvih potprograma [4]:

- Funkcije: vraćaju vrijednost; najviše se koriste za izračun koji mora vratiti nekakvu vrijednost
- Procedure: ne vraćaju direktno vrijednost; najviše se koriste da bi obavili nekakvu akciju

Procedure i funkcije, kao i PL/SQL programi općenito, također imaju strukturu koja se sastoji od 3 glavna bloka, bloka deklaracija, izvršnog bloka i bloka za rukovanje s iznimkama:

```
DECLARE
      <deklaracije>
BEGIN
      <naredbe, izvršni dio programa/procedure>  - jedini obavezan
EXEPCTION
      <rukovanje iznimkama vezanim uz pogreške>
```

Proceduru stvaramo pomoću naredbe CREATE OR REPLACE PROCEDURE. Ovako izgleda općenita sintaksa stvaranja procedure:

```
CREATE [OR REPLACE] PROCEDURE ime_procedure
[(parametar [IN | OUT | IN OUT] tip_podataka [{ := | DEFAULT } izraz],,
...
[(parametar [IN | OUT | IN OUT | tip_podataka)] {IS | AS}
BEGIN
<tijelo_procedure>
END ime_procedure;
```

Procedura koja se stvara na ovakav način kasnije se može pozvati iz glavnog programa dok god je prisutna na serveru. Pohranjena se procedura, u bilo kojem PL/SQL programu u kojem je potrebna, poziva referencom na njeno ime. Procedure se najviše koriste da bi izvršile nekakav proces u PL/SQL programu. Procedura može imati ugniježđene blokove te također može biti i definirana u ugniježđenom bloku unutar drugih blokova ili paketa[10]. Pomoću parametara, vrijednosti se mogu prosljeđivati proceduri, kao i dobivati iz procedure. Pri tome se stvarni parametri, unutar glavnog programa iz kojeg se poziva procedura, moraju uključiti u poziv procedure [8].

[10] Poglavlje 2.13. - Paketi

U sljedećem primjeru prikazano je stvaranje jednostavne procedure u zasebnom bloku, a čija je jedina svrha ispis poruke 'Hello World!'. Naredba 'CREATE PROCEDURE' navodi kompajler na stvaranje nove procedure. Dodatak 'OR REPLACE' omogućuje da nova procedura zamijeni staru istoga imena, ukoliko takva procedura postoji. Pri imenovanju procedure treba paziti da joj ime bude jedinstveno. Ukoliko je procedura ugniježđena unutar drugih blokova koristimo ključnu riječ 'IS', a ukoliko se stvara u zasebnom vlastitom bloku koristimo ključnu riječ 'AS', pri čemu je dakle jedina razlika gdje u kodu stvaramo proceduru, osim toga 'IS' i 'AS' imaju istu funkciju [8].

```
CREATE OR REPLACE PROCEDURE pozdrav
AS

BEGIN

   dbms_output.put_line('Hello World!');

END;
/

Procedure created.
```

Primjer 20. Stvaranje jednostavne procedure u zasebnom bloku

Jednom pohranjenu proceduru moguće je iz glavnoga programa pozvati ili pomoću ključne riječi EXEC (ili EXECUTE) ili korištenjem zasebnog PL/SQL bloka, u oba slučaja referirajući se na ime koje smo dodijelili proceduri, kako slijedi (primjeri se odnose na prethodno stvorenu proceduru *pozdrav*) [4]:

```
EXEC pozdrav;
```

ILI

```
BEGIN
      pozdrav;
END
```

Pohranjenu proceduru brišemo pomoću naredbe DROP PROCEDURE, također referirajući se na njeno ime. Konkretno za proceduru iz prethodnih primjera:

```
DROP PROCEDURE pozdrav;
```

Parametri u PL/SQL-u jesu varijable ili *placeholderi* bilo kojeg tipa podataka važećeg u PL/SQL-u. Pomoću parametara vrši se komunikacija, odnosno razmjena vrijednosti između glavnog programa i potprograma, odnosno procedura ili funkcija. Parametri, dakle, omogućuju davanje ulaza potprogramima, kao i izvlačenje vrijednosti iz potprograma [8].

Parametri se moraju definirati zajedno sa potprogramima uz koje će ih se vezati, odmah kada se potprogram stvara. Tu ih nazivamo formalnim parametrima. U glavnom programu mjesto formalnih parametara zauzimaju stvarni parametri koje prosljeđujemo potprogramu te oni prenose konkretnu vrijednost na mjesto formalnih parametara. Naravno, pri tome se tipovi podataka formalnih i stvarnih parametara moraju poklapati [8].

U PL/SQL-u, dakle, razlikujemo formalne i stvarne parametre. Formalni parametri su kontejner varijable i definiramo ih u definiciji funkcije ili procedure, a stvarni su parametri konkretni parametri koji se iz glavnog programa prosljeđuju pohranjenoj proceduri ili funkciji.

- Formalni parametri imaju *modove* koji opisuju njihovo ponašanje u definiciji funkcije ili pohranjene procedure. Ti su modovi [1]:

- **IN**: definira formalni parametar kao ulaznu varijablu za pohranjeni potprogram, *read-only*, ne može mu se dodijeliti vrijednost unutar tijela potprograma, može biti samo konstanta, ukoliko nije specificirano drukčije, svaki je parametar po defaultu tipa IN
- **OUT**: definira formalni parametar kao kontejner za vraćanje vrijednosti u odgovarajući stvarni parametar, vrijednost mu se može dodijeliti samo u tijelu potprograma, mora biti varijabla
- **IN OUT**: definira formalni parametar kao kontejner varijablu koja se može koristiti i unutar izraza i za vraćanje vrijednosti u odgovarajući stvarni parametar, mora biti varijabla

U Primjeru 22 prikazano je korištenje ovih modeova u proceduri. Brojevi a i b kao formalni parametri procedure imaju *mode* IN jer im je vrijednost konstantna te se ne mijenja unutar potprograma, odnosno same procedure. Oni će tek u glavnom programu poprimiti vrijednost stvarnih parametara, ali ni tada se njihova vrijednost više neće mijenjati.

Broj c ono je što procedura računa, njegova se vrijednost mijenja unutar same procedure, jer ovisi o parametrima a i b, odnosno stvarni parametar z poprimiti će vrijednost onoga koji je od njih manji (što se odvija unutar same procedure) te zato ima *mode* OUT.

```
DECLARE
   x INT;
   y INT;
   z INT;
PROCEDURE Minimum(a IN INT, b IN INT, c OUT INT) IS
BEGIN
   IF a<b THEN
c:= a;
   ELSE
c:= b;
   END IF;
END;

BEGIN
x:= 18;
y:= 25;
Minimum(x, y, z);
   dbms_output.put_line(' Minimum brojeva ' || x || ' i ' || y || ' je : ' || z);

END;
/

Minimum brojeva 18 i 25 je : 18
```

Primjer 21. Procedura s IN i OUT modeovima parametara

2.12. Funkcije

Funkcije su vrlo slične procedurama, s tom razlikom da vraćaju vrijednost. To je ujedno i glavna razlika između funkcija i procedura; pohranjene procedure obavljaju neku akciju, a funkcije vrše izračun na temelju kojeg vraćaju vrijednost [1]. Sintaksa za stvaranje funkcije gotovo je identična onoj za stvaranje procedure, uz dodatak klauzule RETURN, radi vrijednosti koju funkcija mora vratiti. Tip podataka te vrijednosti definira se u CREATE dijelu sintakse [4]:

```
CREATE [OR REPLACE] FUNCTION ime_funkcije
[(parametar [IN | OUT | IN OUT] tip_podataka [{ := | DEFAULT } izraz],,
...
[(parametar [IN | OUT | IN OUT | tip_podataka)] {IS | AS}
RETURN povratni_tip_podataka
{ IS | AS }
< tijelo_funkcije >
```

Funkcije su zasebni PL/SQL blokovi koji se uglavnom koriste za izračune. Vraćanje vrijednosti za funkcije je obavezno; funkcija će ili vratiti vrijednost ili javiti grešku. Funkcije koje ne provode nikakve operacije nad samom bazom podataka mogu se pozvati i direktno preko SQL naredbe SELECT (kao dio SELECT upita). S druge strane, funkcije koje vrše nekakvu radnju nad bazom podataka mogu se pozivati samo iz drugih PL/SQL blokova. Kao i procedure, funkcije mogu sadržavati ugniježđene blokove ili biti ugniježđene unutar drugih blokova ili paketa. Pomoću parametara, vrijednosti se mogu prosljeđivati funkciji, kao i dobivati iz funkcije. To znači da funkcija može vratiti vrijednost i pomoću parametra tipa OUT, a ne samo uz ključnu riječ RETURN [8].

Funkcije se sastoje od dva dijela: tijelo opisa (*specification body*) i tijelo funkcije (*function body*) [1]. U tijelu opisa nužno je definirati povratnu vrijednost kao i sve varijable te kursore koje će funkcija koristiti, a u tijelu funkcije definiramo izračun koji funkcija vrši nad zadanim varijablama.

Ukoliko već postoji funkcija koju želimo modificirati ili napraviti potpuno novu funkciju istoga imena, naredba CREATE OR REPLACE u jednom će koraku zamijeniti staru funkciju i stvoriti novu s istim imenom. S druge strane, naredba CREATE FUNCTION samo stvara novu funkciju, te je potrebno prethodno odbaciti staru funkciju naredbom DROP FUNCTION [1].

Primjer 22 predstavlja funkciju nad tablicom ALBUM koja će na temelju vrijednosti u stupcu 'ukupna cijena' izračunati sveukupnu sumu cijena svih albuma u tablici. Za stvaranje funkcije vrijede ista pravila kao kod procedure; uz dodatak vezan za tip podataka povratne vrijednosti. Dakle, uz ključnu riječ 'RETURN' potrebno je odmah definirati kojeg će tipa podataka biti vrijednost koju funkcija vraća. Kao i kod procedure, ime funkcije mora biti jedinstveno, ključna riječ 'IS' koristi se kada je funkcija ugniježđena unutar drugih blokova, a ključna riječ 'AS' kada se funkcija stvara u vlastitom zasebnom bloku. Budući da ova funkcija ne radi nikakve konkretne promjene nad samom bazom podataka, odnosno ne mijenja njen sadržaj, možemo ju pozvati i uz ključnu riječ SELECT i u zasebnom bloku glavnog programa.

Pomoću kursora 'albumi' smo pospremili sve vrijednosti stupca 'ukupna cijena' iz tablice ALBUM. Osim tog kursora, u opisu funkcije definirali smo i varijablu 'ukupno' tipa podataka INT što je vrijednost koju će funkcija vraćati. Unutar izvršnog bloka (od ključne riječi BEGIN do ključne riječi END) definirano je tijelo funkcije, odnosno konkretan izračun koji će funkcija izvršiti. Uz pomoć navedenog kursora i FOR petlje kursora, s ovom funkcijom računamo sveukupnu cijenu svih albuma u tablici. Na kraju imamo i dvije varijante poziva funkcije te vidimo što funkcija vraća.

```
CREATE OR REPLACE FUNCTION CIJENE_ALBUMA

RETURN INT
AS
UKUPNO INT := 0;

CURSOR albumi IS
      SELECT UKUPNA_CIJENA FROM ALBUM;

BEGIN
      FOR redak in albumi
      LOOP
      UKUPNO := UKUPNO + redak.UKUPNA_CIJENA;
      END LOOP;

      RETURN UKUPNO;
END;
/

Function created.

-- poziv funkcije (SQL naredbom)
SELECT CIJENE_ALBUMA FROM ALBUM;

CIJENE_ALBUMA
-------------
         1334
         1334
...
15 rows selected.

-- poziv funkcije (u zasebnom bloku)
DECLARE
      funkcija varchar2(20);
BEGIN
      Funkcija := CIJENE_ALBUMA;
      dbms_output.put_line(CIJENE_ALBUMA);
END;
/

1334

PL/SQL procedure successfully completed.
```

Primjer 22. Funkcija

2.13. Paketi

Paketi u PL/SQL-u su objekti koji grupiraju logički povezane tipove podataka, varijable i potprograme u jednu novu zasebnu cjelinu. Oni također spadaju u imenovane PL/SQL blokove te se nakon stvaranja kompajliraju i spremaju na server kao objekti baze podataka koji se kasnije mogu koristiti više puta, dok god se nalaze na serveru. Svaki paket sastoji se od dva obavezna dijela [4]:

- Opis (specifikacija) paketa
- Tijelo ili definicija paketa

Specifikacija paketa sadrži deklaracije svih varijabli, kursora, objekata, procedura, funkcija i iznimki koje su dio paketa. Svi objekti koji su dio specifikacije su PUBLIC objekti, što znači da im se može pristupiti izvan paketa. Specifikacija paketa zaseban je element koji može postojati i sama za sebe, bez tijela paketa. Kad god se paket poziva, stvara se nova instanca paketa za tu sesiju. Tada se svi elementi paketa uvode u tu instancu te vrijede do kraja sesije [8].

Specifikaciju paketa stvaramo pomoću naredbe CREATE PACKAGE. Kao i kod funkcija i procedura, varijantom CREATE OR REPLACE automatski ćemo zamijeniti stari paket istoga imena, ukoliko takav postoji. U nastavku je isječak koda koji demonstrira stvaranje specifikacije jednostavnog paketa koji sadrži samo jednu proceduru. Svakom paketu moramo dati proizvoljno, jedinstveno ime, u ovom primjeru 'paket_1'. Nakon ključne riječi 'IS' slijede deklaracije svih PUBLIC objekata koji su dio paketa, a postupak zaključujemo sa END 'ime_paketa'. Ukoliko je postupak stvaranja specifikacije paketa prošao uspješno, program vraća poruku 'Package created'.

```
CREATE OR REPLACE PACKAGE paket_1 IS

   PROCEDURE provjeri_zanr(c_ID ALBUM.ID_ALBUMA%type);

END paket_1;
/

Package created.
```

Primjer 23. Paket 1 – Specifikacija (opis)

Tijelo paketa sadrži definicije svih elemenata koji se nalaze u specifikaciji paketa. Također može sadržavati i definicije elemenata kojih nema u specifikaciji, to su PRIVATE objekti kojima se može pristupiti samo unutar paketa. U tijelu paketa moraju biti definicije svih potprograma i kursora deklariranih u specifikaciji paketa. Za razliku od specifikacije, koja može postojati i zasebno, tijelo paketa postoji isključivo u paru sa specifikacijom te direktno ovisi o njoj. Svaki put kada se kompajlira specifikacija paketa, tijelo paketa postaje nevažeće, što znači da se i ono mora ponovno kompajlirati svaki puta nakon kompajliranja specifikacije. Sve PRIVATE objekte potrebno je definirati prije upotrebe u tijelu paketa. Tijelo paketa ima dva dijela. U prvom dijelu su globalne deklaracije, odnosno varijable, kursori i privatni elementi vidljivi cijelom paketu. Drugi dio sadrži inicijalizaciju paketa, te se izvrši jednom na svako prvo pozivanje paketa u nekoj sesiji [8].

Za stvaranje tijela paketa koristimo naredbu CREATE PACKAGE BODY. Sljedeći primjer demonstrira stvaranje tijela paketa koji sadrži samo jednu proceduru, a čiju smo specifikaciju vidjeli u prethodnom primjeru. U specifikaciji paketa imali smo deklaraciju procedure 'provjeri_zanr', a sada u tijelu paketa moramo definirati tu istu proceduru. Ukoliko je tijelo paketa uspješno stvoreno, program vraća poruku 'Package body created'.

```
CREATE OR REPLACE PACKAGE BODY paket_1 IS

   PROCEDURE provjeri_zanr (c_ID ALBUM.ID_ALBUMA%TYPE) IS
   c_zanr ALBUM.ZANR%TYPE;

      BEGIN
      SELECT ZANR INTO c_zanr
      FROM ALBUM
      WHERE ID_ALBUMA = c_ID;
      dbms_output.put_line('Zanr: '|| c_zanr);
      END provjeri_zanr;

END paket_1;
/

Package body created.
```

Primjer 24. Paket 1 – Tijelo

U nastavku imamo nešto složeniji primjer paketa. Najprije u specifikaciji paketa deklariramo procedure za dodavanje, brisanje i ispis članova benda iz tablice ČLAN BENDA:

```
CREATE OR REPLACE PACKAGE paket_2 IS

   -- Dodavanje clana benda
   PROCEDURE dodajClana(c_ID CLAN_BENDA.ID_CLANA%type,
   c_ID_Benda CLAN_BENDA.ID_BENDA%type,
   c_Ime CLAN_BENDA.IME%type,
   c_Prezime CLAN_BENDA.PREZIME%type,
   c_Uloga CLAN_BENDA.ULOGA%type);

   -- Brisanje clana
   PROCEDURE brisiClana(c_ID CLAN_BENDA.ID_CLANA%TYPE);
   -- Ispis svih clanova
   PROCEDURE ispisClanova;

END paket_2;
/

Package created.
```

Primjer 25. Paket 2 – Specifikacija (opis)

Zatim u tijelu paketa imamo i definicije istih procedura za dodavanje, brisanje i ispis podataka iz tablice ČLAN BENDA.

```
CREATE OR REPLACE PACKAGE BODY paket_2IS

   PROCEDURE dodajClana (c_ID CLAN_BENDA.ID_CLANA%type,
   c_ID_Benda CLAN_BENDA.ID_BENDA%type,
   c_Ime CLAN_BENDA.IME%type,
   c_Prezime CLAN_BENDA.PREZIME%type,
   c_Uloga CLAN_BENDA.ULOGA%type)
   IS

   BEGIN
      INSERT INTO CLAN_BENDA (ID_CLANA, ID_BENDA, IME, PREZIME, ULOGA)
         VALUES(c_ID, c_ID_Benda, c_Ime, c_Prezime, c_Uloga);

   END dodajClana;

   PROCEDURE brisiClana(c_ID CLAN_BENDA.ID_CLANA%type) IS
   BEGIN
      DELETE FROM CLAN_BENDA WHERE ID_CLANA = c_ID;

   END brisiClana;

   PROCEDURE ispisClanova IS

CURSOR c_clanovi IS
      SELECT  IME FROM CLAN_BENDA;
TYPE c_ispis IS TABLE OF CLAN_BENDA.Ime%type;
Ime_ispis c_ispis := c_ispis();
brojacINTEGER :=0;

BEGIN

      FOR redak IN c_clanovi LOOP
brojac := brojac +1;
Ime_ispis.extend;
Ime_ispis(brojac) := redak.Ime;
      dbms_output.put_line('Clan Benda(' || brojac || ')'||Ime_ispis(brojac));
      END LOOP;

   END ispisClanova;

END paket_2;
/

Package body created.
```

Primjer 26. Paket 2 – Tijelo

3. Zaključak

Ovaj završni rad bio je zamišljen kao svojevrstan *tutorial*, odnosno priručnik kroz neke od mogućnosti programskog jezika PL/SQL, proceduralnog proširenja SQL-a. U nekoliko uvodnih poglavlja ukratko su pokrivene osnove samog jezika kao i općeniti oblik njegove sintakse.

U glavnom dijelu, kroz praktične primjere, ponajprije nad vlastitom relacijskom bazom podataka sastavljenom od 5 tablica (3 jednostavne tablice i 2 bez jednostavnog samostalnog primarnog ključa) obrađene su neke od naprednijih mogućnosti PL/SQL-a. PL/SQL pruža veliki broj mogućnosti za uređivanje, ažuriranje, dodavanje i brisanje podataka u SQL bazama podataka. Prosječnom korisniku itekako pojednostavljuje rad s SQL bazama podataka. Nadalje, modularan pristup koji PL/SQL koristi također olakšava i održavanje i pisanje koda. Osnovna struktura svakog PL/SQL programa od 3 bloka koji dalje mogu imati svoje podblokove poprilično je intuitivna i kroz neko kraće vrijeme korištenja vrlo ju je jednostavno shvatiti i primjenjivati na ispravan način.

U konačnici, sve što je obrađeno u ovom završnom radu tek je mali uvid u sve ono što može PL/SQL, koji ima još mnogo mogućnosti, standardnih paketa, koncepata i sličnog. Radi se o nadasve zanimljivom i moćnom programskom jeziku koji je u kombinaciji s SQL-om jako koristan alat za rad s relacijskim bazama podataka.

Po mojem osobnom dojmu PL/SQL je uistinu jednostavan, intuitivan, koristan, *user-friendly* programski jezik vrlo sličan drugim proceduralnim programskim jezicima. Uz osnovno predznanje SQL-a te općenite logike proceduralnih programskih jezika, vrlo ga je lako savladati do razine koja će biti i više nego zadovoljavajuća za prosječnog korisnika. Naravno, da bi ga se naučilo u dubinu ipak je potrebno uložiti nešto više vremena i truda radi iznimno velikog broja mogućnosti koje PL/SQL pruža.

PL/SQL na tržištu se dosta koristi u kombinaciji s SQL-om i kao takav je svakako bolja opcija za rad s relacijskim bazama podataka od korištenja isključivo SQL-a. Međutim, PL/SQL je ipak vrlo specifičan programski jezik sa posebnom namjenom pisanja koda unutar Oracle baza podataka. Kao takav je izvrstan, ali njegove mogućnosti ipak su vezane uz mali broj određenih zadataka, te izvan Oracle baza podataka PL/SQL i nema široku namjenu. Također, sve organizacije koje rade s relacijskim bazama podataka ne mogu si nužno priuštiti Oracle baze podataka što je još jedan minus za PL/SQL.

Konsenzus informatičke zajednice je da PL/SQL ne napreduje niti raste na tržištu, ali također se ne očekuje niti njegov skori nestanak s tržišta. Njegova je popularnost i upotreba trenutno u dugom periodu stagnacije. Ipak, radi popularnosti i široke upotrebe Oracle baza podataka uz koje je PL/SQL usko vezan, predviđa se da će još dugo imati svoje mjesto na tržištu.

Što se tiče PL/SQL programera, radi iznimne specifičnosti PL/SQL-a kao programskog jezika vezanog isključivo uz Oracle baze podataka te činjenice da će se u toj branši najvjerojatnije još dugo koristiti, kroz nekoliko godina predviđa se porast plaća i poboljšanje uvjeta za PL/SQL programere zato što na neki način postaju „ugrožena vrsta“. Sve je manje novih, mladih programera koji savršeno barataju PL/SQL-om, a iskusni PL/SQL stručnjaci s vremenom sve više prelaze na druge, bolje plaćene pozicije. Ostaje nam vidjeti koliko će se sva ova predviđanja pokazati točnima.

Literatura

[1] McCullough-Dieter Carol, Prem Jatinder, Chandak Ramesh, Chandak Purshottam, Oracle 8 biblija. Znak, Zagreb, 1998.

[2] Kursori Baze Podataka u Oracle 11g, Istratech. Preuzeto 01.08.2018. sa http://www.istratech.hr/wp-content/uploads/2008/11/case2008.pdf

[3] PL/SQL, Wikipedia. Preuzeto 24.07.2018. sa https://en.wikipedia.org/wiki/PL/SQL

[4] PL/SQL Tutorial, Tutorialspoint. Preuzeto 01.08.2018. sa https://www.tutorialspoint.com/plsql/

[5] M. Bobrowski Steven, Oracle 7 i obrada podataka po modelu klijent/server. Mikro knjiga, Beograd, 1995.

[6] Rob Peter, Coronel Carlos, Database Systems: Design, Implementation & Management. Course Technology, Boston, 2004.

[7] Pavlić Mile, Oblikovanje Baza Podataka. Odjel za informatiku sveučilišta u Rijeci, Rijeka, 2011.

[8] PL/SQL Tutorial, Guru99. Preuzeto 12.09.2018. sa https://www.guru99.com/pl-sql-tutorials.html

Popis tablica

Popis slika

Popis primjera koda