Priručnik za pogramiranje u Oracle PL/SQL-u

Puriš, Mia

Sveučilište u Rijeci – Fakultet informatike i digitalnih tehnologija

Informacijski i komunikacijski sustavi

Mia Puriš

Priručnik za programiranje u Oracle PL/SQL-u

Diplomski rad

Mentor: prof. dr. sc. Patrizia Poščić

Rijeka, rujan 2022.

ODJEL ZA INFORMATIKU
SVEUČILIŠTE U RIJECI

Rijeka, 10. lipnja 2021.

Zadatak za diplomski rad

Pristupnik: Mia Puriš

Naziv diplomskog rada: Priručnik za programiranje u Oracle PL/SQL-u

Naziv diplomskog rada na eng. jeziku: Oracle PL/SQL programming tutorial

Sadržaj zadatka:

Zadatak diplomskog rada je napisati priručnik za učenje programiranja u PL/SQL-u. Cilj je odgovoriti na neka temeljna pitanja poput što je PL/SQL uopće, koje su mu funkcionalnosti, kako postaviti okruženje, sintaksa (tipovi podataka, petlje, funkcije, kursori itd…). Potrebno je definirati i izraditi vlastiti primjer modela entiteti-veze nad kojim bi se objašnjavala sintaksa PL/SQL jezika te na kojem bi se temeljio cijeli priručnik.

Mentor:

Prof. dr.sc. Patrizia Poščić

Voditeljica za diplomske radove:

Izv. prof. dr. sc. Ana Meštrović

Komentor:

Zadatak preuzet: 10. lipnja 2021.

(potpis pristupnika)

Zahvaljujem se svojoj obitelji, tati, mami i sestrama, što su uvijek uz mene, i što me podupiru čak i onda kada padam i kada se sve čini nemogućim.

Najviše se zahvaljujem svojim sestrama Vlatki i Tei koje su bile moj najveći kamen oslonac i koje su me bodrile i bile moje najveće navijačice. One koje su u bilo koje doba dana ili noći nalazile potrebne riječi koje su me držale da krenem dalje.

Želim se zahvaliti svim kolegama i prijateljima koji će mi ostati u prekrasnom sjećanju na ovaj fakultet, a posebno Sanji Jotić i Dini Periću koji su za svaki moj problem našli rješenje i bili potpora u najtežim trenucima. Hvala im na tome.

Potom se zahvaljujem i svojoj mentorici, bez koje sve ovo ne bi bilo moguće.

Per aspera ad astra.

Sadržaj

1. Sažetak

Oracle Corporation pruža moćne sustave za upravljanje relacijskim bazama podataka. Za bolje razumijevanje i za maksimalno iskorištavanje potencijala koje Oracle nudi, potrebno je razumjeti na koji način PL/SQL može manipulirati podacima.

Glavna tema rada je izrada priručnika za Oracle PL/SQL programski jezik odnosno dan je pregled PL/SQL-a zajedno sa njegovim značajkama, prednostima ali i razlikama između SQL-a i PL/SQL-a. Unutar rada svaka teorijska cjelina potkrepljena je praktičnim primjerima. Rad je podijeljen na jedanaest glavnih cjelina.

Korištena je relacijska baza podataka za krojačnicu koju je autorica koristila za svoj završni rad na preddiplomskom studiju. Baza će služiti za objašnjavanje mogućnosti koje nudi PL/SQL. Za kreiranje baze korišten je Oracle 19c sustav za upravljanje bazom podataka, dok je za pisanje koda i rad s njom korišten Oracle SQL Developer.

Ključne riječi: PL/SQL, SQL, Oracle, model podataka, krojačnica, relacijske baze podataka

2. Uvod

2.1 Ukratko o PL/SQL-u

PL/SQL (eng. *Procedural Language Extensions to Structured Query Language*) proceduralni je jezik relacijske baze podataka[1]. SQL (eng. *Structured Query Language*) sveprisutan je jezik za upite i ažuriranje relacijskih baza podataka. Oracle je krajem 1980-ih razvio PL/SQL kako bi uklonio neka od ograničenja unutar SQL-a, odnosno PL/SQL proceduralno je proširenje SQL-a s proceduralnim osobinama poput deklaracije varijabli, petlji, iteracije i tako dalje. PL/SQL pruža Oracle bazu podataka koja je ugrađena u programsko okruženje neovisno o operacijskom sustavu. Omogućuje jednostavniji razvoj aplikacija, optimizaciju izvršenja i poboljšano korištenje resursa baze podataka. Opća sintaksa PL/SQL-a temelji se na programskom jeziku Ada i Pascal te sam programski jezik nije osjetljiv na velika i mala slova kao i SQL.

Unatoč svojoj međusobnoj povezanosti, SQL i PL/SQL dva su potpuno različita jezika. SQL je ograničeni jezik koji omogućuje izravnu interakciju s bazom podataka. On omogućuje manipuliranje objektima (eng. *Data definition language*[2], skraćeno DDL) i podacima (eng. *Data manipulation language*[3], skraćeno DML), ali nedostaju mu neke značajke koje su prisutne u drugim programskim jezicima poput petlji i IF-THEN naredbi. To je svrha PL/SQL-a. PL/SQL standardni je jezik sa svim značajkama koje se nalaze u većini drugih programskih jezika.

2.2 PL/SQL u klijent/poslužitelj arhitekturi

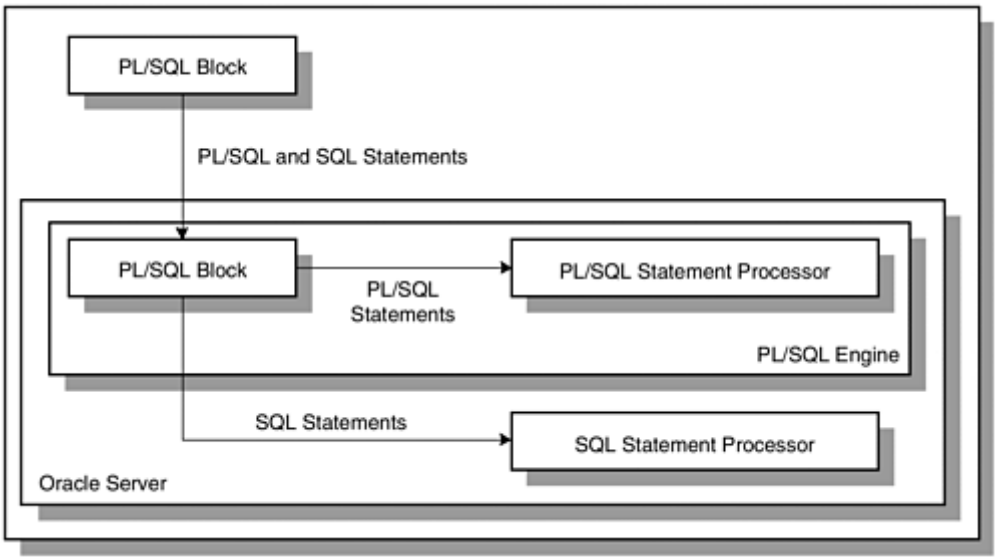
Klijent/poslužitelj arhitektura koristi se u razvoju mnogih Oracle aplikacija. Oracle baza podataka učitana je na poslužitelja. Program koji postavlja zahtjeve prema bazi podataka

[1] Relacijska baza podataka je vrsta baze podataka koja pohranjuje i pruža pristup podacima koji su međusobno povezani. Temelji se na relacijskom modelu, koji je jednostavan i intuitivan način predstavljanja podataka u tablicama. Svaki red u relacijskoj bazi podataka zapis je s jedinstvenim identifikatorom koji se naziva ključ. Stupci tablice sadrže atribute podataka, a svaki zapis obično ima vrijednost za svaki atribut što olakšava uspostavljanje odnosa između podataka [4] [6] [50].

[2] Sintaksa za stvaranje i modificiranje objekata baze podataka poput tablica, indeksa, korisnika. Neke od DDL naredbi su CREATE, ALTER, DROP.

[3] Programski jezik koji se koristi za dodavanje, brisanje i ažuriranje podataka u bazi podataka. Neke od DML naredbi su SELECT, INSERT, UPDATE, DELETE.

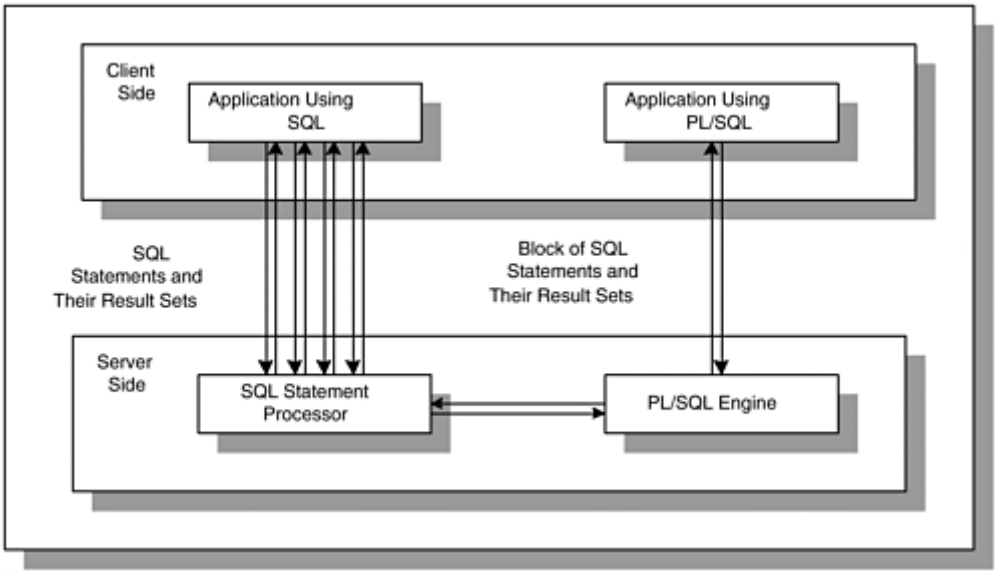
nalazi se na stroju klijenta. Potrebno je naglasiti da PL/SQL nije samostalan programski jezik (eng. *Standalone programming langugage*) nego je komponenta Oracle RDBMS-a (eng. *Relational database management system*) i može se nalaziti u dva okruženja, klijentu i poslužitelju. Rezultat toga je jednostavnije premještanje PL/SQL modula između aplikacija na strani klijenta i poslužitelja. PL/SQL stroj (eng. *PL/SQL Engine*), koji je posebna komponenta mnogih Oracle proizvoda (Oracle server, Oracle Forms, Oracle Reports i tako dalje), obrađuje bilo koji PL/SQL blok ili potprogram u oba okruženja. SQL i PL/SQL razlikuju se u i načinu izvođenja. Sve PL/SQL naredbe obrađuju se i izvršavaju od strane PL/SQL stroja, a sve SQL naredbe šalju se na procesor koji je odgovoran za obradu SQL naredbi (eng. *SQL Statement Processor*). Slika 1 prikazuje obradu u slučaju kada se PL/SQL stroj nalazi na poslužitelju.



Slika 1 PL/SQL stroj i Oracle poslužitelj - obrada PL/SQL bloka [1]

U slučaju kada se PL/SQL stroj nalazi na strani klijenta, u tom trenutku i sama obrada odvija se na strani klijenta. Sve SQL naredbe koje su unutar PL/SQL bloka prosljeđuju se na poslužitelja na daljnju obradu.

Jedna od značajnijih prednosti PL/SQL-a je sposobnost grupiranja više SQL naredbi u pojedinačan upit, koji se šalje na poslužitelja gdje ih pritom poslužitelj i izvršava. Nakon izvršenja svake naredba, rezultat se vraća nazad korisniku. Kada se šalje više naredbi, one se šalju kao jedna cjelina te je rezultat čitavog bloka isto tako jedna cjelina. Taj proces učinkovitiji je nego da se svaka naredba izvršava pojedinačno [1]. Slika 2 prikazuje komunikaciju između klijenta i poslužitelja.



Slika 2 Prikaz komunikacije između klijenta i poslužitelja - klijent/poslužitelj arhitektura [1]

2.3 Prednosti PL/SQL-a

Uz prethodno spomenute prednosti, postoje još neke [2] [3]:

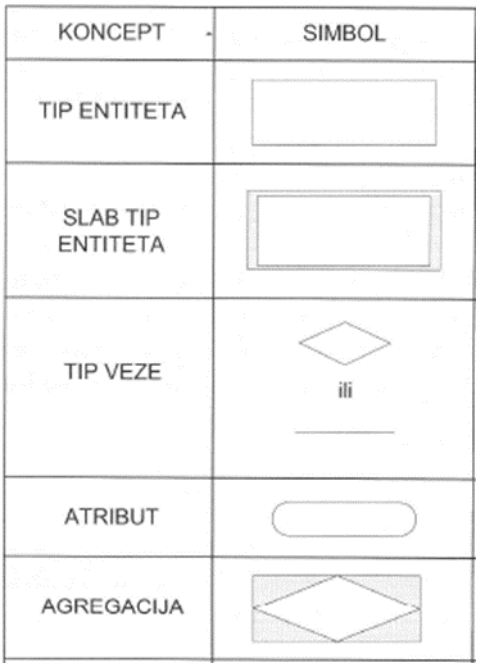
- Smanjenje mrežnog prometa prilikom izvršavanja cijelog bloka naredbi
- Prilikom izvršavanja PL/SQL program učinkovito obrađuje sve pogreške i prikazuje poruke o greškama koje su prilagođene korisniku za rješavanje problema
- Bolja izvedba kada se SQL naredbe izvršavaju u bloku nego pojedinačno
- Povezanost s SQL-om
- Blokovi koda mogu se ugnijezditi jedan unutar drugoga i pohraniti u bazu podataka i kasnije ponovno koristiti
- Sadrži velik broj tipova podataka
- Podržava objektno orijentirano programiranje
- Prenosivost aplikacije odnosno aplikacije koje su napisane u PL/SQL-u mogu se prenijeti u bilo koji operativni sustav i mogu raditi neovisno na bilo kojem računalu
- Podržava strukturirano programiranje kroz funkcije i procedure
- Podržava razvoj web aplikacija
- Sadrži velik broj predefiniranih SQL paketa

3. Model podataka

Model podataka (eng. *Data Model*) skup je međusobno povezanih podataka koji opisuju entitete, veze i atribute poslovnog sustava. On je reprezentacija skupa podataka koji se modelom interpretiraju preko aspekta: strukture, ograničenja i operatora [4] [5].

Sastoji se od:

- **Entitet** (eng. *entity*) – realni ili konceptualni element sustava, to je neka posebnost što u poslovnome sustavu postoji i jasno se razlikuje od drugih entiteta
- **Veza** – (eng. *relationship*) – koncept koji predstavlja neku interakciju među entitetima u sustavu, odnosno predstavlja znanje o njihovoj povezanosti
- **Atribut entiteta** (eng. *attribute*) – neko svojstvo entiteta
- **Agregacija** (eng. *aggregation*) – apstrakcija gdje se skup tipova entiteta i njihovih veza predstavlja novim izvedenim tipom entiteta
- **Veza** (eng. *relationship*) – pridruživanje između entiteta



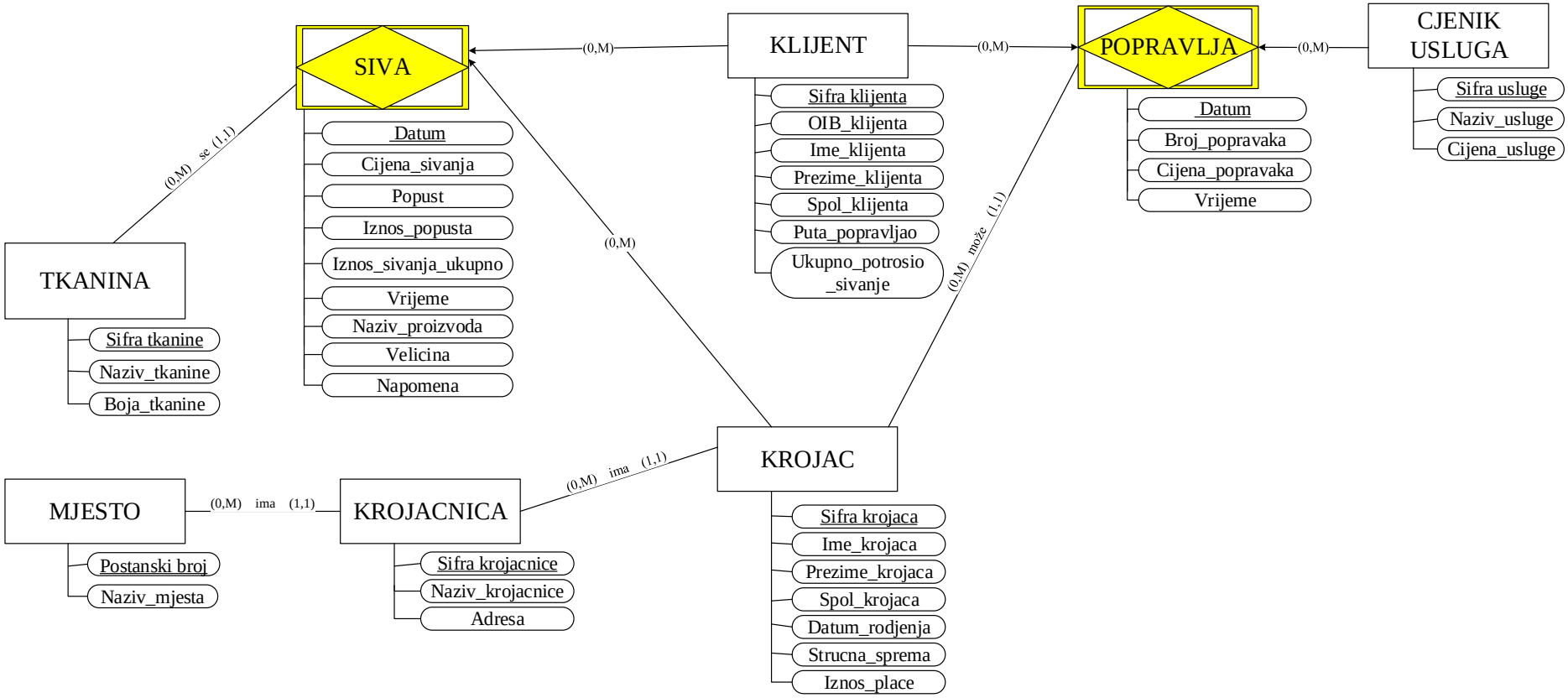
Slika 3 Prikaz grafičkih simbola modela podataka [4]

Pomoću modela podataka ilustriraju se datoteke koje su organizirane u relacijskoj bazi podataka. Model je napravljen u alatu Microsoft Visio i korištena je specijalizirana metodologija MIRIS[4]. U dijagramu entitet veza (EV) nalazi se 8 tipova entiteta (odnosno tablica, slika 4) od kojih su dvije slabe agregacije (SIVA i POPRAVLJA). Svakom entitetu pridruženi su odgovarajući atributi, primarni ključevi (koji su podcrtani punom linijom). Vanjski ključevi zastupljeniji su u relacijskom modelu. U EV dijagramu vanjski ključevi iščitavaju se u ovisnosti brojnosti veze, na način da tip entiteta koji ima na svojoj strani GG 1 (najviše 1) sadrži VK te veze (primjer je tablica KROJACNICA koja ima polje Postanski_broj za vezu KROJACNICA - MJESTO). Kod veze slabe agregacije s jakim tipom entiteta (primjer SIVA – KLIJENT – KROJAC i POPRAVLJA – KLIJET – CJENIK_USKLUGA) nije potrebno pisati naziv i brojnost veze – na strani slabe agregacije brojnost je GG 1 (najviše 1), a na strani jačeg tipa entiteta je brojnost GG M (najviše više). Nije potrebno pisati primarne ključeve koji su uzeti iz jakih tablica (kod slučaja slabog tipa entiteta primarni ključ dobiva od jakog tipa entiteta, te sadrži i svoj vlastiti primarni ključ). U slučaju agregacije, ona dobiva primarni ključ iz onih tablica gdje je brojnost M:M. Ali također slabe tipove entiteta i agregacije moguće je povezati i s drugim tablicama u modelu (u ovom slučaju to su tablice KROJAC i POPRAVLJA) i ta veza piše se kao obična i doda joj se još naziv i brojnost.

U EV dijagramu svaka KROJACNICA ima sjedište u samo jednom MJESTU (DG najmanje 1 i GG najviše 1), a svako MJESTO može sadržavati više KROJACNICA (DG najmanje 0, GG najviše M). Svaka KROJACNICA može imati više KROJACA (DG najmanje 0 i GG najviše 1) dok se svaki KROJAC nalazi u samo jednoj KROJACNICI (DG najmanje 1 i GG najviše 1). Svaki KROJAC može SIVATI više TKANINA (DG najmanje 0 i GG najviše 1), a svaku TKANINU može SIVATI više KROJACA (DG najmanje 0 i GG najviše M) te SIVA ne može prethodno postojati bez da prethodno ne postoje tablice KROJAC i KLIJENT (agregacija, zavisna tablica). Svaki KROJAC može više puta POPRAVLJATI (DG najmanje 0 i GG najviše M), a POPRAVLJANJE može biti samo od jednog KROJACA (DG najmanje 1 i GG najviše 1). Svako POPRAVLJANJE ide samo po jednom CJENIKU_USLUGA (DG najmanje 1 i GG najviše 1), dok svaki CJENIK_USLUGA može imati više cijena POPRAVLJANJA (DG najmanje 1 i GG najviše M) te POPRAVLJA ne može prethodno postojati bez da prethodno ne postoje tablice KLIJENT i CJENIK_USLUGA (agregacija,

[4] Specijalizirana metodologija MIRIS (skraćeno: Metodologija za Razvoj Informacijskog sustava) skup je metoda i uputa čiji je ukupni cilj projektirati i izgraditi informacijski sustav. Ta specijalizirana metodologija propisuje faze razvoja i aktivnosti pojedine faze do potrebne razine detalja informacijskih sustava. [4]

zavisna tablica). Svaki KLIJENT može POPRAVLJATI više stvari koje su navedene u CJENIKU_USLUGA (DG najmanje 0 i GG najviše M), a svaka CIJENA_USLUGE može biti kod više KLIJENATA (DG najmanje 0 i GG najviše M). Jednom KLIJENTU može se SIVATI nešto od više TKANINA (DG najmanje 0 i GG najviše M). Jedna TKANINA može biti korištena u SIVANJU za više KLIJENATA (DG najmanje 1 i GG najviše M) [5].



Slika 4 Prikaz relacijskog modela za poslovnu aplikaciju Krojačnica

4. Relacijski model

U relacijskom modelu podataka prikazani su svi tipovi entiteta iz EV dijagrama, zajedno sa svim svojim atributima, uključujući i PK (podcrtano punom linijom) i VK (podcrtano isprekidanom linijom). Svaki tip entiteta može imati samo jedan PK (jednostavan ili složen od više atributa) te može imati više VK (jednostavnih ili složenih – potencijalno po jedan VK za svaku vezu) [6].

MJESTO (Postanski broj, Naziv_mjesta)

TKANINA (Sifra tkanine, Naziv_tkanine, Boja_tkanine)

KROJACNICA (Sifra krojacnice, Naziv_krojacnice, Adresa, Postanski broj)

KLIJENT (Sifra klijenta, OIB_klijenta, Ime_klijenta, Prezime_klijenta, Spol_klijenta, Puta_popravljao, Ukupno_potrosio_sivanje)

CJENIK USLUGA (Sifra usluge, Naziv_usluge, Cijena_usluge)

SIVA (Sifra klijenta, Sifra krojaca, Datum, Cijena_sivanja, Popust, Iznos_popusta, Iznos_sivanja_ukupno, Vrijeme, Naziv_proizvoda, Velicina, Napomena, Sifra tkanine)

KROJAC (Sifra krojaca, Ime_krojaca, Prezime_krojaca, Spol_krojaca, Datum_rodjenja, Strucna_sprema, Iznos_place, Sifra krojacnice)

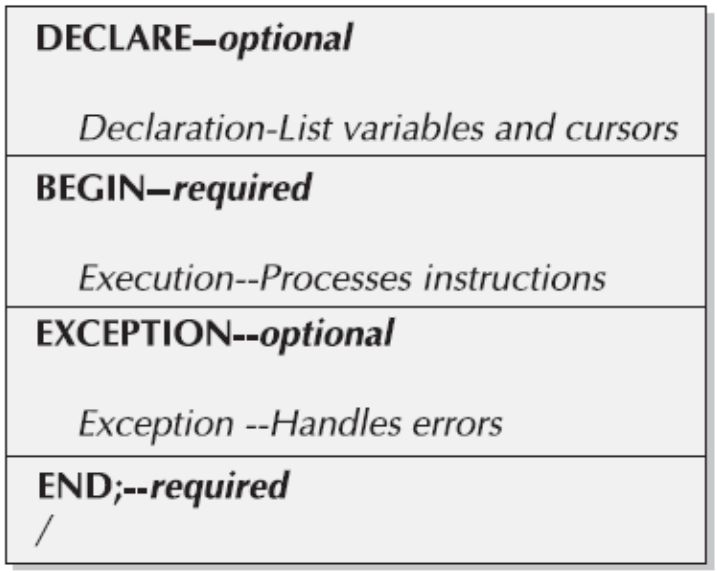
POPRAVLJA (Sifra klijenta, Sifra usluge, Datum, Broj_popravaka, Cijena_popravaka, Vrijeme, Sifra krojaca)

5. Osnove PL/SQL jezika

5.1 Struktura PL/SQL bloka

PL/SQL je blok-strukturirani jezik (eng. *Block Structured Language*) čiji je kod organiziran u blokove. Blok je osnovna programska jedinica unutar PL/SQL-a. Blokovi se mogu ugnijezditi jedan unutar drugoga. Oni uglavnom kombiniraju izraze koji predstavljaju jedan logički zadatak, te je stoga moguće da se različiti zadaci unutar jednog programa mogu podijeliti u blokove. Blokovi se dijele na imenovane i anonimne (neimenovani). Imenovani blokovi (eng. *Named blocks*) koriste se prilikom stvaranja potprograma te oni posjeduju zaglavlje (eng. *Header*). Primjer takvih blokova su procedure, funkcije i paketi koji se mogu pohraniti u bazu podataka te ih se kasnije može ponovno pozvati. Anonimni blokovi (eng. *Anonymous blocks*) ne mogu se spremiti u bazu podataka, odnosno mogu se koristiti samo jednom. Zbog toga su korisni za testiranje. Drugim riječima ukoliko zaglavlje nije određeno, blok se smatra anonimnim. PL/SQL blok čine tri dijela: deklarativni dio, izvršni dio i dio za obradu iznimaka. Izvršni dio jedini je obavezni dio bloka. Minimalna struktura za PL/SQL blok su rezervirane riječi **BEGIN** i **END** s najmanje jednom izvršnom naredbom između njih. Kosa crta (eng. *Slash*) izvršava anonimni PL/SQL blok [7] [8].

```
BEGIN
        NULL;
END
/
```



Slika 5 Prikaz strukture PL/SQL bloka [9]

5.1.1 Deklarativni dio

Unutar deklarativnog dijela (eng. *The Declaration Section*) definiraju se konstante, varijable, kursori, korisnički definirane iznimke i ostali elementi koji će se koristiti u kodu. Deklarativni dio nije obavezan dio bloka te se može preskočiti ukoliko nije potrebno definirati varijable ili druge elemente programa. Deklarativni dio počinje ključnom riječju **DECLARE** za okidače i anonimni blok. Prije samog ispisa na ekran, potrebo je promijeniti vrijednost varijable okruženja SERVEROUTPUT, koja je po definiciji postavljena na OFF.

```
SET SERVEROUTPUT ON;
DECLARE
    sifra_klijenta NUMERIC;
    ime_klijenta VARCHAR(30);
```

5.1.2 Izvršni dio

Kako je prethodno spomenuto, izvršni dio (eng. *The Execution Section*) jedini je obavezni dio bloka. Sadrži naredbe koje će PL/SQL stroj izvršiti tijekom izvođenja, odnosno može sadržavati dodjelu varijabli, inicijalizaciju objekata, uvjetne strukture, iterativne strukture, ugniježđene anonimne PL/SQL blokove i tako dalje. Izvršni dio počinje ključnom riječju **BEGIN** i završava ključnom riječju **END**.

Primjer:

```
DECLARE
-- definiranje varijabli x i y
-- tip podataka je numeričke vrijednosti
    x NUMBER;
    y NUMBER;
BEGIN
    x:= 5;
    y:= 4;
    DBMS_OUTPUT.PUT_LINE ('Rezultat množenja:  ' ||  x * y);
END;
/
```

Rezultat:

Rezultat množenja: 20

PL/SQL procedure successfully completed.

Izvršni dio treba sadržavati barem jednu naredbu, čak i ako je to **NULL** naredba koja ne radi ništa. Podržava sve DML naredbe i SQL*Plus ugrađene funkcije. DDL naredbe podržava pomoću nativnog dinamičkog SQL-a (eng. *Native Dynamic SQL*[5], skraćeno NDS) i/ili DBMS_SQL[6] ugrađenog paketa.

5.1.3 Dio za obradu iznimaka

Dio za obradu iznimaka (eng. *The Exception Section*) posljednji je dio bloka koji također nije obavezan. Sadrži naredbe koje omogućuju presretanje i obradu posebnih uvjeta koji bi se mogli dogoditi tijekom izvođenja programa (na primjer dijeljenje s nulom, dupliciranа vrijednost primarnog ključa). Blok može „*uhvatiti*“ određene greške uz pomoć funkcija koje se nalaze unutar STANDARD ili DBMS_STANDARD paketa[7] ili korištenjem naredbe EXCEPTION_INIT Pragma. Dio za obradu iznimaka počinje ključnom riječju **EXCEPTION**.

Primjer koda za model krojačnice:

```
DECLARE
        naziv VARCHAR(30);
BEGIN
        SELECT naziv_mjesta INTO naziv
        FROM mjesto
        WHERE postanski_broj = 21000;
        DBMS_OUTPUT.PUT_LINE ('Naziv mjesta je: ' || naziv);

 EXCEPTION
```

[5] Dopušta aplikaciji izvršavanje SQL naredbi čiji je sadržaj nepoznat dok se ne izvrši. Glavna prednost je što omogućuje izvođenje DDL naredbi koje nisu podržane izravno unutar PL/SQL-a (na primjer stvaranje tablica) [11] .

[6] Pruža sučelje za korištenje dinamičkog SQL-a za izvođenje jezika manipulacije podacima (DML) i jezika definicije podataka (DDL), izvršavanje PL/SQL anonimnih blokova i pozivanje PL/SQL pohranjenih procedura i funkcija [49].

[7] STANDARD i DBMS_STANDARD paketi deklariraju i definiraju osnovne tipove podataka (na primjer VARCHAR2 i NUMBER), kao i potprograme kao što je RAISE APPLICATION ERROR (omogućuje generiranje korisničkih definiranih iznimaka iz bloka ili pohranjenog programa) [51].

```
            WHEN NO_DATA_FOUND THEN
                DBMS_OUTPUT.PUT_LINE ('Ne postoji mjesto s tim poštanskim
brojem!!!');
    END;
    /
```

Rezultat:

Ne postoji mjesto s tim poštanskim brojem!!!

PL/SQL procedure successfully completed.

Sve naredbe završavaju točkom-zarezom bez obzira u kojem se bloku nalaze. SQL naredbe unutar PL/SQL bloka izvršavaju se u izvršnom dijelu gdje se konačan rezultat naredbe zapisuje u varijablu koja je prethodno deklarirana u deklarativnom dijelu bloka.

5.2 Jezične osnove

Kao i svaki drugi programski jezik, PL/SQL također sadrži pravila. PL/SQL programi pisani su u obliku linija teksta koje koriste određeni skup znakova. PL/SQL stroj prihvaća četiri vrste znakova:

- Velika i mala slova: A - Z i a - z
- Brojevi: 0 - 9
- Znakovi: ( ) + - * / < > = ! ~ ^ ; : . ‘ @ % , “ # $ & _ | { } ? [ ]
- Tabulatori, razmak i oznake za potvrdu: Tab, Space, Enter

Spajanjem elemenata iz jedne ili više spomenutih vrsta znakova, stvara se leksička jedinica. U programskim jezicima leksičke jedinice služe kao temelj iz razloga što omogućuju stvaranje PL/SQL blokova. Leksičke jedinice klasificiraju se na:

- Identifikatore
- Literale
- Znakove za odvajanje (jednostavni i složeni)
- Komentare

5.2.1 Identifikatori

Identifikatori se koriste za imenovanje programskih objekata i jedinica koji uključuju konstante, varijable, iznimke, kursore, potprograme, rezervirane riječi i pakete [10]. Nazivi identifikatora sami po sebi trebaju biti razumljivi. Prilikom izrade identifikatora treba obratiti pažnju na sljedeće:

- Maksimalna duljina identifikatora iznosi 30 znakova (znak dolara ($), podvlaka (_) i brojevi također ulaze u konačan zbroj znakova)
- Identifikator se sastoji od bilo kojih znakova sve dok početni znak u nazivu nije $, #, _, ili bilo koji broj
- Ne smije sadržavati interpunkciju, razmake ili crtice
- Prema zadanim postavkama nisu osjetljivi na velika i mala slova (na primjer ne raspoznaje razliku između postanski_broj i Postanski_Broj)

Primjer identifikatora:

Važeći nazivi identifikatora	Nevažeći nazivi identifikatora
Trenutna_Placa	Trenutna-Placa
CijenaUsluge1	1CijenaUsluge
Naziv_proizvoda_#	Naziv+proizvoda
Adresa_#2	Adresa #2
Maksimalna_vrijednost_radnikove_place	max_placaRadnik

Tablica 1 Prikaz važećih i nevažećih identifikatora

Identifikatori u dvostrukim navodnicima (eng. *Quoted Identifiers*): PL/SQL nudi mogućnost stavljanja identifikatora u dvostruke navodnike zbog fleksibilnosti. Kada su identifikatori stavljeni u dvostruke navodnike, mogu sadržavati nestandardne znakove i razmake. Oni razlikuju velika i mala slova te mogu čak sadržavati i rezervirane riječi koje u većini slučajeva nisu dopuštene. Inače se ne preporuča korištenje tih nestandardnih identifikatora (izgledom su slični na literale niza) osim u slučajevima kada su baš prijeko potrebni pošto će uzrokovati probleme.

Primjer: "Odjel za informatiku's"
"Naš/Vaš"
"Rijeka"

Posebni znakovi (eng. *Special Characters*): Posebni znakovi su identifikatori koje PL/SQL razumije kao naredbu. Korištenje ovih znakova unutar PL/SQL-a u druge svrhe osim za što su namijenjeni, rezultirati će greškom ili pogrešnom obradom koda.

Rezervirane riječi (eng. *Reserved Words*): Rezervirane riječi su identifikatori koje je Oracle izdvojio za internu upotrebu (BEGIN, END i SELECT i tako dalje). Ne bi se trebale koristiti kod naziva varijabli, kursora, naziva stupaca, tablica, indeksa, literala ili kod korisnički definirane iznimke.

Primjer:

```
DECLARE
        default VARCHAR2(15);
BEGIN
        default := 'Nemoj koristit rezervirane rijeci za nazive varijabli!';
        DBMS_OUTPUT.PUT_LINE(default);
END;
/
```

Rezultat:

Error report -
ORA-06550: line 2, column 1:
PLS-00103: Encountered the symbol "DEFAULT" when expecting one of the following:

begin function pragma procedure subtype type <an identifier>
<a double-quoted delimited-identifier> current cursor delete exists prior
06550. 00000 - "line %s, column %s:\n%s"
**Cause: Usually a PL/SQL compilation error.*
**Action:*

U primjeru deklarirana je varijabla DEFAULT. Zatim se varijabla inicijalizira i trebala bi se prikazati njezina vrijednost na ekranu. No međutim zbog krivo upotrijebljene rezervirane riječi dogodila se greška zato što se ta riječ ne može koristit kod imenovanja varijable.

5.2.2 Znakovi za odvajanje

Oracle ih koristi za posebnu namjenu. Djeluju kao separatori, matematički operatori (na primjer koriste se za predstavljanje aritmetičkih operacija poput zbrajanja i oduzimanja) i operatori spajanja. Operatori su detaljnije opisani u poglavlju *5.6 Operatori*. Popis znakova za odvajanje prikazan je u tablici 2 [10].

Znak za odvajanje	Opis
+, -, *, /, **	Aritmetički operatori
..	Operator za domet, najčešće se koristi kod FOR petlji
<, >, <>, =, !=, ~=, ^=, <=, >=	Relacijski operatori
--, /*, */	Indikatori komentara
<<, >>	PL/SQL oznaka za odvajanje (početak i kraj)
%	Indikator atributa, koristi se kod atributa TYPE, ROWTYPE, NOTFOUND ali i drugih
:	Indikator varijable povezivanja
,	Separator predmeta
'	Separator niza znakova
"	Separator identifikatora u navodnicima
||	Operator povezivanja
@	Pokazatelj udaljenog pristupa
=>	Operator asocijacije – koristi se prilikom pozivanja procedure ili funkcije te kod prosljeđivanja vrijednosti parametrima
;	Kraj naredbe – koristi se pri završetku svake naredbe ili deklaracije unutar PL/SQL-a
:=	Operator dodjele – inicijalizira varijable (lijevo od operatora) s vrijednostima (desno od operatora)

Tablica 2 Prikaz jednostavnih i složenih znakova za odvajanje [9]

5.2.3 Literali

Literali su vrijednosti koje nisu predstavljene identifikatorom nego uključuju znak, niz, broj, logičke konstante i datum/vrijeme vrijednosti.

Znak (eng. *Charachter*): Znakovni literali pojedinačni su znakovi u jednostrukim navodnicima. Mogu sadržavati bilo koji alfanumerički ili poseban znak te su osjetljivi na velika i mala slova. Brojevi koji se tretiraju kao literali znakova (kada su u navodnim znakovima) ne tretiraju se kao brojevi osim ako se ne koriste u aritmetici. Slični su literalima niza, no razlika je u tome što njihova veličina iznosi jedan znak. Detaljnije su objašnjeni u poglavlju *5.3 Tipovi podataka.*

Primjer: ' M' 'i' '$' ' ' '7' '#' '[' '

Znakovni niz (eng. *String*): Svi alfanumerički znakovi, posebni znakovi i interpunkcija uključeni su u literale niza. Osjetljivi su na velika i mala slova.

Primjer: q'#Sveučilište u Rijeci'10.3.1995' informatika.#','$7,000'

Broj (eng. *Number*): Numerički literali dijele se na cjelobrojne i realne vrijednosti. Cjelobrojne vrijednosti cijeli su brojevi bez decimala, a realne vrijednosti su brojevi s jednom decimalom. Znanstvena notacija može se koristiti za izražavanje cjelobrojnih i realnih vrijednosti. Detaljnije su objašnjeni u poglavlju *5.3 Tipovi podataka*.

Primjer: 10 5.56 3E2 20e-02 .1 1. +12 -6.7 -2.5D -1.5F

Logičke konstante (eng. *Boolean*): TRUE, FALSE i NULL Booleove su vrijednosti dostupne unutar PL/SQL-a. Boolean varijabla može se deklarirati i može joj se dodijeliti literalna vrijednost. Ne nalaze se u navodnicima i nisu nizovi. Kada se kombiniraju s blokom IF-THEN, ne moraju imati usporednu vrijednost ako se uspoređuju samo s TRUE. Ali ako se uspoređuju s FALSE ili NULL, vrijednost mora biti navedena u usporedbi. Detaljnije su objašnjene u poglavlju *5.3 Tipovi podataka.*

Primjer:

```
DECLARE
        vrijednostTrue BOOLEAN := TRUE;
        vrijednostFalse BOOLEAN := FALSE;
        vrijednostNull BOOLEAN := NULL;
BEGIN
       IF vrijednostTrue
       THEN
            DBMS_OUTPUT.PUT_LINE('TRUE');
        END IF;
```

```
        IF vrijednostFalse
        THEN
            DBMS_OUTPUT.PUT_LINE('FALSE');
         END IF;

        IF vrijednostNull
        THEN
            DBMS_OUTPUT.PUT_LINE('NULL');
        END IF;
END;
/
```

Rezultat:

TRUE

PL/SQL procedure successfully completed.

Datum/vrijeme vrijednost (eng. *Date/Time*): Mogu biti predstavljene u nekoliko formata, kao što su nizovi u navodnicima ili kao brojevi. Detaljnije su objašnjene u poglavlju *5.3 Tipovi podataka*. Literali datum/vrijeme imaju četiri različita formata ovisno o tipu podataka:

- DATE
- TIMESTAMP
- TIMESTAMP s vremenskom zonom
- TIMESTAMP s lokalnom vremenskom zonom

Primjer:

```
DECLARE
        vrijednostDate DATE := DATE '2021-11-07';
        vrijednostTimestamp TIMESTAMP := TIMESTAMP '2021-11-07 21:31:43';
        vrijednostTimestampVremenskaZona TIMESTAMP WITH TIME ZONE :=
TIMESTAMP '2021-11-07 21:31:43 +01:00';
        vrijednostTimestampLokalnaVremenskaZona TIMESTAMP WITH LOCAL
TIME ZONE := TIMESTAMP '2021-11-07 21:31:43';
BEGIN
        DBMS_OUTPUT.PUT_LINE(vrijednostDate);
        DBMS_OUTPUT.PUT_LINE(vrijednostTimestamp);
        DBMS_OUTPUT.PUT_LINE(vrijednostTimestampVremenskaZona);
```

```
        DBMS_OUTPUT.PUT_LINE(vrijednostTimestampLokalnaVremenskaZona);
END;
/
```

Rezultat:

```
2021-11-07
07.11.21 21:31:43,000000
07.11.21 21:31:43,000000 +01:00
07.11.21 21:31:43,000000

PL/SQL procedure successfully completed.
```

5.2.4 Komentari

Komentari se koriste kako bi se PL/SQL stroju naznačilo da ono što slijedi treba zanemariti. Komentari se obično koriste da se drugim programerima objasne neka temeljna načela ili logika koda. Postoje dvije vrste komentara unutar PL/SQL-a: komentar u jednoj liniji i komentar u više linija.

Komentar u jednoj liniji: Započinje s dvije povlake (--), koje se ne mogu odvojiti razmakom ili bilo kojim drugim znakom. Prevoditelj (eng. *Compiler*) zanemaruje sav tekst nakon dvije povlake do kraja linije i ta se cijela linija smatra komentarom.

Primjer:

```
DECLARE
-- ovdje se deklariraju varijable, kursori
-- konstante, i tako dalje
        ...
BEGIN
-- ovdje se piše kod
        ...
END;
```

Komentar u više linija: Započinje sa kosom crtom i zvjezdicom (/*), a završava sa zvjezdicom i kosom crtom (*/), čime može obuhvatiti više uzastopnih linija teksta. Poželjno bi bilo da se koristi dovoljno komentara kako bi kod bio jasniji. Može se koristit za:

- Informacije zaglavlja PL/SQL objekta
- Informacije za zaglavlje skripte

- Komentiranje blokova koda. To je korisno kod testiranja ili za učinkovito uklanjanje koda, a da pritom ostane dostupan u slučaju da bude potreban u budućnosti
- Dokumentaciju koda, posebno komentari koji sežu izvan pojedinačnih linija

Primjer:

```
DECLARE
/*  Sveučilište u Rijeci
    Odjel za informatiku */
    ...
BEGIN
    ...
END;
```

5.3 Tipovi podataka (1/2)

Svakoj konstanti ili varijabli pridružen je podatkovni tip koji određuje njen oblik pohranjivanja, ograničenja i valjan raspon vrijednosti. PL/SQL nudi mnoštvo unaprijed definiranih tipova podataka i podtipova te također omogućuje definiranje svojih PL/SQL podtipova. Podtip je podskup drugog tipa podataka, koji se naziva osnovnim tipom. Podtip može izvesti iste operacije kao i njegov osnovni tip, ali se može koristit samo podskup njegovih valjanih vrijednosti. PL/SQL tipovi podataka mogu se podijeliti na četiri glavne kategorije [9]:

- Skalarni (eng. *Scalar*)
- Veliki objekti (eng. *Large Object* – skraćeno LOB)
- Referencirani (eng. *Reference*)
- Kompozitni (eng. *Composite*)

Kompozitni i referencirani tipovi podataka detaljnije su obrađeni u poglavlju *7. Tipovi podataka*.

5.3.1 Skalarni tipovi podataka

To su osnovni tipovi podataka koji pohranjuju jednu vrijednost poput broja ili niza znakova. Dijele se na četiri potkategorije:

- Numerički (eng. *Numeric*)

- Znakovni (eng. *Charachter*)
- Logički (eng. *Boolean*)
- Datumski i vremenski (eng. *Date/Time*)

5.3.1.1 Numerički tipovi podataka

Numerički tipovi podataka predstavljaju realne i cijele brojeve te brojeve s pomičnim zarezom nad kojima se mogu izvoditi aritmetičke operacije [11] [12].

BINARY_FLOAT i **BINARY_DOUBLE**: Tipovi podataka koji predstavljaju brojeve s pomičnim zarezom jednostruke i dvostruke preciznosti prema IEEE 754 standardu. BINARY_FLOAT literal završava na slovo f (na primjer 4.45f) dok BINARY_DOUBLE završava na slovo d (na primjer 5.000065d). BINARY_DOUBLE zauzima 64 bita za zapis, dok BINARY_FLOAT zauzima duplo manje, odnosno 32 bita za zapis. Uglavnom se koriste u znanstvenim izračunima.

PLS_INTEGER i **BINARY_INTEGER**: Ta dva tipa podatka identična su. Pohranjuju cijele brojeve u rasponu od -2 147 483 648 do 2 147 483 647 i zapis zauzima 32 bita. U usporedbi s NUMBER tipom podatka potrebno je manje mjesta za pohranu i njihove operacije izvode se strojno.

NUMBER: Tip podatka koji pohranjuje cijele i decimalne brojeve (brojevi s fiksnim ili pomičnim zarezom s apsolutnim vrijednostima u rasponu od 1E-130 do 1.0E126 (ne uključujući) ali isto tako može poprimiti i vrijednost 0). Prema Oraclu preporuča se korištenje samo NUMBER literala i rezultat NUMBER računanja koji su unutar navedenog raspona. U suprotnome događa se:

- Svaka vrijednost koja je premala zaokružuje se na nulu
- Vrijednost literala koja je prevelika uzrokuje grešku kompilacije
- Rezultat izračuna koji je prevelik je nedefiniran, što uzrokuje nepouzdane rezultate i moguće greške tijekom izvođenja

Sintaksa za određivanje broja s fiksnim zarezom je NUMBER (*preciznost*, *skala*) gdje *preciznost* (eng. *precision*) predstavlja ukupan broj znamenki, a *skala* (eng. *scale*) označava broj znamenki nakon decimalne točke (na primjer NUMBER (6,2)). Sintaksa za određivanje

cijelog broja je NUMBER (*preciznost*). Na primjer NUMBER (4) označava da je ukupan broj znamenaka četiri. Za brojeve s pomičnim zarezom, decimalni zarez može se zapisati na bilo koju poziciju. Što se tiče sintakse dovoljno je samo napisati NUMBER. Maksimalna vrijednost parametra *preciznost* iznosi 38, a za parametar *skala* minimalne i maksimalne vrijednosti su između brojeva -84 i 127. Na primjer ako vrijednost parametra *skale* iznosi 2, tada se vrijednost 7.232 zaokružuje na najbližu deseticu, odnosno na 7.23. No ako je vrijednost parametra *skale* poprimilo negativnu vrijednost, odnosno -1, vrijednost 7232 zaokružuje se na najbližu deseticu s lijeve strane decimalne točke, to jest zaokružuje se na vrijednost 7230. Ako zadnja vrijednost parametra *skale* iznosi 0, decimalni brojevi zaokružuju se na najbliži cijeli broj to jest vrijednost 7.232 zaokružuje se na broj 7 [11] [12].

5.3.1.2 Znakovni tipovi podataka

Znakovni tipovi podataka omogućuju pohranu i izmjenu (spajanje, rastavljanje i tako dalje) alfanumeričkih vrijednosti koje predstavljaju pojedinačne znakove ili nizove znakova [11].

CHAR i **VARCHAR**: Nizovi znakova fiksne i promjenjive duljine pohranjeni su u ta dva tipa podataka. Svi literali niza imaju CHAR tip podatka. Sintaksa za određivanje CHAR i VARCHAR tipa podatka je [CHAR | VARCHAR2] *[(maksimalna_duljina* [CHAR | BYTE])], gdje *maksimalna_duljina* mora biti cjelobrojni literal u rasponu od 1 do 32 767, a ne konstanta ili varijabla. Ukoliko parametar duljine nije određen, najveća duljina iznosi jedan znak. Inicijalizacijski parametar NLS_LENGTH_SEMANTICS određuje zadanu jedinicu veličine (CHAR ili BYTE). Najveća dopuštena duljina znakovnog niza je do 32 767 bajta, bez obzira da li se za *maksimalna_duljina* navede veličina u znakovima ili bajtovima.

RAW: Tip podatka koji pohranjuje binarne nizove ili nizove bajtova poput niza grafičkih znakova ili digitalizirane slike. RAW podaci slični su VARCHAR2 podacima, s razlikom da PL/SQL ne tumači neobrađene podatke. Sintaksa za određivanje RAW tipa podatka je RAW (*maksimalna_duljina*). *maksimalna_duljina* mora biti cjelobrojni literal u rasponu od 1 do 32 767, a ne konstanta ili varijabla. Ukoliko parametar duljine nije određen, najveća duljina iznosi jedan znak. Nije moguće unijeti RAW varijablu s vrijednošću većom od 2 000 bajtova u stupac baze podataka istog tipa.

NCHAR i **NVARCHAR2**: Tipovi podataka koji pohranjuju nacionalne nizove znakova (eng. *National character strings*)[8] fiksne i promjenjive duljine. NCHAR identičan je CHAR tipu podataka te se njihove vrijednosti mogu zamijeniti u naredbama ali treba uzeti u obzir da je sigurnije pretvoriti CHAR vrijednost u NCHAR vrijednost nego obrnuto upravo iz razloga što može doći do gubitka podataka. Ista situacija je i za NVARCHAR2 koji je identičan tipu VARCHAR2. Sintaksa za određivanje ta dva tip podatka je NCHAR [(*maksimalna_duljina*)], odnosno NVARCHAR2 (*maksimalna_duljina*). *maksimalna_duljina* mora biti cjelobrojni literal, a ne konstanta ili varijabla. Predstavlja najveći broj znakova, a ne najveći broj bajtova koji iznosi 32 767. Najveća NCHAR vrijednost koja se može umetnuti u stupac NCHAR baze podataka je 2 000 bajtova, dok kod NVARCHAR2 iznosi 4 000 bajtova.

LONG i **LONG RAW:** LONG tip podatka pohranjuje nizove znakova promjenjive duljine. Sličan je tipu podatka VARCHAR2, s razlikom da najveća duljina LONG vrijednosti iznosi 32 760 bajta. Može pohraniti tekst, niz znakova te kratke dokumente. LONG RAW tip podatka pohranjuje binarne ili bajtne nizove. LONG RAW podaci slični su LONG podacima. Najveća dopuštena duljina također iznosi 32 760 bajtova. Za oba tipa podatka najveća dopuštena širina LONG ili LONG RAW stupca baze podataka iznosi 2 147 483 648 bajtova odnosno 2 GB, može se umetnuti bilo koja LONG vrijednost u LONG stupac i bilo koja LONG RAW vrijednost u LONG RAW stupac. Vrijednost veća od 32 760 bajtova ne može se dohvatit iz LONG stupca u varijablu LONG ili iz LONG RAW stupca u LONG RAW varijablu.

ROWID i **UROWID**: Svaka tablica baze podataka ima ROWID pseudo stupac koji pohranjuje binarne vrijednosti *rowid* te predstavlja adresu pohrane retka, odnosno identificira redak u običnoj tablici i na taj način ubrzava pristup njima. Tip podataka ROWID može sadržavati samo fizičke *rowidove* ali tip podatka UROWID može pohraniti fizičke, logičke i strane *rowidove*. Oracle preporučuje korištenje UROWID tipa podatka kada je to god moguće. Pomoću funkcije ROWIDTOCHAR koja pretvara binarnu vrijednost u niz znakova veličine 18 bajta, dohvaća se varijabla ROWID. Postoji i funkcija CHARTOROWID koja pretvara niz znakova ROWID u *rowid*.

[8] U Unicode bazi podataka, nacionalni niz znakova je niz bajtova koji predstavljaju znakovne podatke u UTF-8 ili UTF-16BE kodiranju. Broj kodnih jedinica u nizu određuje duljinu niza. Ako je duljina nula, vrijednost se naziva prazan niz. Ovu vrijednost ne treba miješati s vrijednošću NULL [52].

5.3.1.3 Logički tip podataka

Logički tip podataka, odnosno BOOLEAN sprema tri logičke vrijednosti TRUE, FALSE i ne vrijednost NULL. Prilikom dodjeljivanja jedne od te tri vrijednosti, nije potrebno koristiti navodnike zato što će u suprotnome doći do greške:

Primjer:

```
DECLARE
        varijabla_boolean BOOLEAN;
BEGIN
        varijabla_boolean := 'FALSE';
END;
/
```

Rezultat:

```
 Error report -
ORA-06550: line 4, column 26:
PLS-00382: expression is of wrong type
ORA-06550: line 4, column 5:
PL/SQL: Statement ignored
06550. 00000 -  "line %s, column %s:\n%s"
*Cause:    Usually a PL/SQL compilation error.
*Action:
```

Sintaksa za određivanje BOOLEAN tipa podataka je:

varijabla_null BOOLEAN; -- implicitno je dodijeljena NULL vrijednost
varijabla_true BOOLEAN NOT NULL := TRUE; -- eksplicitno je dodijeljena vrijednost TRUE
varijabla_false BOOLEAN NOT NULL := FALSE; -- eksplicitno je dodijeljena vrijednost FALSE

Preporuka je da bi se BOOLEAN varijable uvijek trebale eksplicitno inicijalizirati u svojim programskim jedinicama. NOT NULL klauzula osigurava da BOOLEAN varijable tijekom deklaracije nikad nisu NULL. Pošto SQL nema tip podataka koji je ekvivalentan BOOLEAN tipu podatka, stoga se ne može koristit u SQL naredbama, ugrađenim SQL funkcijama (na primjer TO_DATE) te u PL/SQL funkcijama koje se pozivaju iz SQL naredbi. BOOLEAN vrijednosti najčešće se koriste u kontrolnoj strukturi IF – THEN, CASE i u petljama kao što su LOOP, FOR LOOP i WHILE LOOP [11] [13].

5.3.1.4 Datumski i vremenski tip podataka

Tipovi podataka koji omogućuju pohranjivanje i manipulaciju s datumima, vremenima i intervalima (razdoblja). Postoje tri glavna tipa podataka: DATE, TIMESTAMP i INTERVAL.

DATE: Koristi se za pohranjivanje datuma određene duljine (stoljeće, godina, mjesec, dan, sat, minuta i sekunda). Ugrađene funkcije TO_DATE i TO_CHAR koriste se za pretvaranje znakovnih tipova podataka ili datuma u niz (string). Valjani datumi su od 1. siječnja 4712. godine prije Krista do 31. prosinca 9999. godine. NLS_DATE_FORMAT Oracleov je parametar inicijalizacije koji postavlja zadani format datuma. Na primjer zadani format može biti oblika 'DD-MON-YY', što znači da se sastoji od dvoznamenkastog broja za dan u mjesecu, kratice naziva mjeseca dok posljednje dvije znamenke predstavljaju godinu (na primjer 15-NOV-21.). SYSDATE je funkcija koja vraća trenutni datum i vrijeme. Kada treba izdvojiti datum iz vremenske oznake, koristi se poziv funkcije TRUNC(*varijabla_datuma*). To je korisno kada treba naprimjer locirati sve transakcije koje su se dogodile na određeni dan. Također datumi se mogu dodavati i oduzimati. PL/SQL interpretira cjelobrojne literale kao dane u aritmetičkim izrazima. Na primjer SYSDATE + 1 označava isto vrijeme ali sutradan.

TIMESTAMP: Proširuje tip podataka DATE, pohranjuje godinu, mjesec, dan, sat, minutu i sekundu. Sintaksa glasi TIMESTAMP[(*preciznost*), gdje parametar *preciznost* (eng. *precision*) označava duljinu broja koji se zapisuje poslije sekundi i koji predstavlja djelić sekunde koji je protekao u zadanoj preciznosti. Simbolička konstanta ili varijabla ne može se koristiti za određivanje preciznosti nego se moraju koristiti cjelobrojni literali u rasponu od 0 do 9. Zadana vrijednost je šest.

Primjer:

```
DECLARE
        varijablaDatetime TIMESTAMP (6) := SYSTIMESTAMP;
BEGIN
         DBMS_OUTPUT.PUT_LINE(varijablaDatetime);
END;
/
```

Rezultat:

15.11.21 19:39:31,037000

PL/SQL procedure successfully completed.

TIMESTAMP WITH TIME ZONE: Proširuje tip podataka TIMESTAMP. Razlika je u pomaku vremenske zone (satima i minutama između lokalnog vremena i univerzalnog kodiranog vremena (skraćeno UTC) koje je ranije bilo srednje vrijeme po Greenwichu (skraćeno GMT)). Sintaksa je slična kao i kod TIMESTAMPA, TIMESTAMP[(*preciznost*)] WITH TIME ZONE.

Primjer:

```
DECLARE
    varijablaDatetime TIMESTAMP (4) WITH TIME ZONE :=
SYSTIMESTAMP;
BEGIN
    DBMS_OUTPUT.PUT_LINE(varijablaDatetime);
END;
/
```

Rezultat:

15.11.21 19:51:26,4640 +01:00

PL/SQL procedure successfully completed.

TIMESTAMP WITH LOCAL TIME ZONE: Proširuje tip podataka TIMESTAMP WITH TIME ZONE. Razlika je u tome što vraća vrijeme koje odgovara lokaciji klijenta koji pristupa poslužitelju podataka. Na primjer ako klijent iz Zagreba pristupa poslužitelju baze podataka koji se nalazi u Los Angelesu u 20:08 PST[9], TIMESTAMP WITH LOCAL TIME ZONE prikazao bi da je 11:08 sati (Zagreb je 9 sati ispred lokacije Los Angeles), što odgovara postavkama vremenske zone klijenta.

Primjer:

```
DECLARE
    varijablaDatetime TIMESTAMP (0) WITH LOCAL TIME ZONE :=
    SYSTIMESTAMP;
BEGIN
```

[9] Pacific Time Zone

```
    DBMS_OUTPUT.PUT_LINE(varijablaDatetime);
END;
/
```

Rezultat:

15.11.21 20:15:30

PL/SQL procedure successfully completed.

INTERVAL YEAR TO MONTH: Tip podatka koji sprema i upravlja intervalima koji predstavljaju godine i mjesece. Sintaksa glasi INTERVAL YEAR[(*preciznost*)] TO MONTH, gdje parametar *preciznost* (eng. *precision*) određuje preciznost godina. Simbolička konstanta ili varijabla ne može se koristiti za određivanje preciznosti nego se moraju koristiti cjelobrojni literali u rasponu od 0 do 4. Zadana vrijednost je dva.

Primjer:

```
DECLARE
    varijablaIntervalYM  INTERVAL YEAR(3) TO MONTH;
BEGIN
    varijablaIntervalYM := INTERVAL '2-4' YEAR TO MONTH;  -- Interval
    literal
    DBMS_OUTPUT.PUT_LINE('Vremenski period: ' || varijablaIntervalYM);
    varijablaIntervalYM := '3-9';  -- Implicitna pretvorba iz znakovnog tipa
    DBMS_OUTPUT.PUT_LINE('Vremenski period: ' || varijablaIntervalYM);
    varijablaIntervalYM := INTERVAL '12' YEAR;  -- Navedene su samo godine
    DBMS_OUTPUT.PUT_LINE('Vremenski period: ' || varijablaIntervalYM);
    varijablaIntervalYM := INTERVAL '8' MONTH;  -- Navedeni su samo
    mjeseci
    DBMS_OUTPUT.PUT_LINE('Vremenski period: ' || varijablaIntervalYM);
END;
/
```

Rezultat*:*

Vremenski period: +002-04
Vremenski period: +003-09
Vremenski period: +012-00
Vremenski period: +000-08

PL/SQL procedure successfully completed.

Primjer prikazuje da se na više načina može dodijeliti vrijednost varijabli tipa podatka INTERVAL YEAR TO MONTH. Vrijednost se može zapisati kao literal, kako je i zadano prema sintaksi. Ali isto tako moguće je navesti samo godine ili samo mjesece ili samo dodati vrijednost za godinu i mjesec bez navedenog tipa podatka.

INTERVAL DAY TO SECOND: Tip podatka koji sprema i upravlja intervalima koji predstavljaju dane, sate, minute i sekunde. Sintaksa glasi INTERVAL DAY [(*preciznost_dana*) TO SECOND(*preciznost_sekundi*), gdje *preciznost_dana* (eng. *leading_precision*) i *preciznost_sekundi* (eng. *fractional_seconds_precision*) predstavljaju preciznost za dan i dio sekunde. Simbolička konstanta ili varijabla ne može se koristiti za određivanje preciznosti nego se moraju koristiti cjelobrojni literali u rasponu od 0 do 9. Zadane vrijednosti su dva (za dan) odnosno šest (za dio sekunde).

Primjer:

```
DECLARE
        varijabla_intervalDS  INTERVAL DAY(3) TO SECOND(5);
BEGIN
        varijabla_intervalDS := INTERVAL '111 19:46:15' DAY TO SECOND;
        DBMS_OUTPUT.PUT_LINE('Vremenski period: ' ||varijabla_intervalDS);
        varijabla_intervalDS := varijabla_intervalDS + '7 07:07:07';
         DBMS_OUTPUT.PUT_LINE('Vremenski period: ' ||varijabla_intervalDS);
        varijabla_intervalDS := varijabla_intervalDS + INTERVAL '3' MINUTE;
        DBMS_OUTPUT.PUT_LINE('Vremenski period: ' ||varijabla_intervalDS);
END;
/
```

Rezultat:

```
Vremenski period: +111 19:46:15.00000
Vremenski period: +119 02:53:22.00000
Vremenski period: +119 02:56:22.00000

PL/SQL procedure successfully completed.
```

Kako je prikazano i u prethodnom primjeru, svaka vrijednost može se dodati zasebno.

5.3.2 Veliki objekti

Veliki objekti (eng. *Large Object*) sami po sebi nisu tip podataka nego su klasifikacija tipova podataka koji obrađuju velike objekte (pohrana i manipulacija velikih blokova nestrukturiranih podataka poput slika, teksta, multimedijskih datoteka i tako dalje). Postoje četiri različita LOB tipa podataka koji se dijele u dvije skupine: interni (BLOB, CLOB, NCLOB) i vanjski (BFILE) veliki objekti. Interni i vanjski LOB-ovi imaju LOB lokatore i LOB vrijednosti, ali njima upravljaju na različite načine. Oracle daje prednost LOB tipu podataka u odnosu na LONG iz razloga što je fleksibilniji. Prednosti su:

- Tablica u kojoj je tip podataka LONG može imati najviše jedan stupac, dok kod tablice kod koje je tip podataka LOB nema ograničenja na broj stupaca
- LONG stupce potrebno je ručno duplicirati
- LOB tip podataka stalno se poboljšava u svakoj Oracle verziji kako bi zadovoljio suvremene zahtjeve, dok je LONG tip podataka konstantan i ne dobiva mnogo ažuriranja
- Veličina LONG stupca iznosi 2 GB, dok LOB stupac može pohraniti do 128 TB

5.3.2.1 LOB lokatori

LOB lokator (eng. *LOB locator*) prisutan je u svakom LOB tipu podataka. Kada se umetne interna LOB vrijednost veća od 4K, ona se ne pohranjuje izravno u tablicu, odnosno vanjski LOB-ovi nikad se izravno ne pohranjuju u tablicu. Umjesto toga umeće se lokator ili referenca na fizičko mjesto pohrane. Oracle koristi LOB lokator za dohvaćanje ispravne LOB vrijednosti kada je to potrebno [9].

5.3.2.2 Interni LOB

Interni LOB ili LOB čijom pohranom upravlja baza podataka, pohranjen je kao dio tablice ili kao tablični prostor baze podataka s LOB lokatorom pohranjenim u tablici. LOB vrijednost i LOB lokator prisutni su u internoj LOB instanci. Interne LOB instance koje su

pohranjene u tablici smatraju se trajnim (eng. *Persistant*) LOB-ovima[10]. Interni LOB-ovi također mogu biti privremeni (eng. *Temporary*), što znači da LOB instanca postoji samo u kontekstu privremene sesije ili aplikacije. Privremeni prostor tablice koristi se za pohranjivanje LOB vrijednosti. Instanca postaje trajna ako se pohrani u tablicu, u suprotnome LOB instanca se briše kada se instanca oslobodi ili kada sesija završi. Tijekom DML operacija Oracle kopira LOB lokator i LOB vrijednost, bez obzira na to da li je LOB instanca trajna ili privremena [14] [9].

BLOB: Binarni veliki objekt (eng. *Binary Large Object*, skaćeno BLOB) je tip podataka za čitanje i upisivanje odnosno može pohraniti binarne objekte poput slika, videa ili audio datoteka. BLOB tipovi podataka tretiraju se drugačije od skalarnih tipova podataka. Imaju tri moguća stanja, a to su NULL, prazno i popunjeno stanje. BLOB-ovi mogu pohraniti binarne datoteke veličine između 8 do 32TB. PL/SQL omogućuje da se lokalne BLOB varijable mogu deklarirati u anonimnim i imenovanim blokovima. Pošto BLOB-ovi pripadaju u interne velike objekte, oni se pohranjuju i njima se upravlja u bazi podataka. BLOB varijabla čija je vrijednost NULL, deklarira se na sljedeći način:

varijabla BLOB;

Prazna i varijabla koja sadrži vrijednost deklariraju se na sljedeći način:

varijablaPrazna BLOB := empty_blob();
varijabaVrijednost BLOB := '55'||'51'||'41';

CLOB: Veliki objekt znakova (eng. *Character Large Object,* skraćeno CLOB) koristi se za pohranu i čitanje znakova i djeluje kao BLOB tip podataka za tekstualne nizove. CLOB-ovi mogu biti u jednom od tri stanja, a to su NULL, prazno i popunjeno stanje. CLOB varijabla čija je vrijednost NULL, deklarira se na sljedeći način:

varijabla CLOB;

Prazna i varijabla koja sadrži vrijednost deklariraju se na sljedeći način:

[10] Trajni (eng. *Persistent*) LOB je LOB instanca koja je pohranjena u retku tablice baze podataka. Trajni LOB-ovi sudjeluju u transakcijama baze podataka. U slučaju neuspjeha transakcije ili medija, trajni LOB-ovi mogu se oporaviti, a sve promjene trajne LOB vrijednosti mogu se predati ili poništiti. Drugim riječima svojstva nedjeljivosti transakcije (eng. *Atomicity*), konzistentnosti (eng. *Consistency*), izolacije (eng. *Isolation*) i izdržljivosti (eng. *Durability*) koja se primjenjuju na objekte baze podataka primjenjuju se i na trajne LOB-ove [16].

varijablaPrazna CLOB := empty_clob();
varijabaVrijednost CLOB := 'MIA';

NCLOB: Veliki objekt u Nacionalnom skupu znakova (eng. *National Character Set Large Object,* skraćeno NCLOB) koristi se za pohranu i čitanje Unicode znakova[11]. NCLOB tipovi podataka ponašaju se slično kao i CLOB tipovi podataka, s iznimkom da im je dodijeljeno više prostora zato što koriste Unicode skup znakova. Mogu sadržavati do 4GB znakovnih podataka.

5.3.2.2 Vanjski veliki objekti

Operacijski sustav pohranjuje i upravlja vanjskim LOB-ovima izvan baze podataka. Iz tog razloga ograničeno je što se može raditi s datotekama unutar baze podataka, budući da njihovo upravljanje kontrolira operacijski sustav. Oracle ne može osigurati konzistentnost čitanja datoteke, a operacije sigurnosnog kopiranja/oporavka moraju eksplicitno uključivati datoteke vanjskog LOB operativnog sustava koje nisu uključene prema zadanim postavkama. Budući da je vrijednost LOB-a vanjska u odnosu na bazu podataka, Oracle kopira samo LOB lokator prilikom umetanja vanjskih LOB-ova. Kako se kopira samo referenca ili lokator to se naziva referentnom semantikom (eng. *Reference semantics*) [15] [16].

BFILE: Vanjska binarna datoteka (eng. *External Binary File,* skraćeno BFILE) je objekt podataka koji su pohranjeni u datotekama operacijskog sustava, izvan tabličnih prostora baze podataka. Podaci koji su pohranjeni u stupcu tablice tipa BFILE, fizički se nalaze u datoteci operacijskog sustava, a ne u bazi podataka. Za razliku od drugih LOB tipova podataka, BFILE je dozvoljen samo za čitanje (osim za postavljanje virtualnog direktorija i naziva datoteke za vanjsku datoteku). Unutar BFILE-a ne može se pisati niti se može ažurirati unutar programa. BFILE obično se koristi za pohranu:

- Binarnih podataka koji se ne mijenjaju za vrijeme pokrenute aplikacije
- Podataka koji se učitavaju u druge tipove podataka (na primjer BLOB ili CLOB) te se nad njima vrši obrada

[11] Unicode je univerzalni kodirani skup znakova koji omogućuje pohranjivanje informacija s bilo kojeg jezika pomoću jednog skupa znakova. Pruža jedinstvenu vrijednost koda za svaki znak, bez obzira na platformu, program ili jezik. [53]

- Podataka koji su prikladni za pristup *byte streamu* (na primjer multimedija)

Svaki uređaj za pohranu (na primjer tvrdi disk, CD-ROM, DVD-ROM) kojemu može pristupiti operacijski sustav, može sadržavati BFILE podatke.

5.4 Deklariranje varijabli i konstanti

Varijable unutar PL/SQL-a moraju se deklarirati kao globalna varijabla unutar deklarativnog dijela bilo kojeg bloka, potprograma ili paketa. Za pridruživanje vrijednosti varijablama koristi se operator dodjele "**:=**" ili se koristi rezervirana riječ DEFAULT. Svaka deklaracija završava s točkom-zarez. Ako se varijabla deklarira kao konstanta, ona će zadržati istu vrijednost u cijelome bloku s time da joj se prvotno mora dati vrijednost prilikom deklaracije. Kada se varijabla deklarira, PL/SQL dodjeljuje memorijski prostor za pohranjivanje njezine vrijednosti. Prilikom izvršenja programa varijable se smatraju promjenjivim veličinama, a konstante nepromjenjivima. Sintaksa za deklaraciju varijabli je:

ime_varijable [CONSTANT] *tip_podataka* [NOT NULL] [:= DEFAULT *inicijalna_vrijednost*];

ime_varijable je ime koje se dodjeljuje varijabli i može se navesti da li je varijabla konstanta na način da se poslije imena varijable doda ključna riječ CONSTANT. *tip_podataka* označava kojem će tipu podataka varijabla pripadati. U slučaju kada je varijabla konstanta, dodaje joj se inicijalna vrijednost koja se u bloku ne može mijenjati. Inicijalna vrijednost koju joj PL/SQL dodijeli je NULL. Ukoliko se za varijablu odredi da njezina vrijednost ne bude NULL, to se postiže dodavanjem ograničenja NOT NULL nakon što se varijabli odredi tip podatka [1] [17].

Primjer:

```
DECLARE
    sifra_krojaca NUMBER(4);
    ime_krojaca VARCHAR(30);
    prezime_krojaca VARCHAR(30) := 'Horvat';
    iznos_place CONSTANT DECIMAL(5,2) := 3450.00;
```

Kako je prethodno spomenuto, PL/SQL dopušta da se blokovi ugnijezde jedan unutar drugoga, odnosno svaki programski blok može sadržavati još jedan unutarnji blok. U tom slučaju treba pripaziti na vidljivost varijabli koja se dijeli se na **lokalnu** i **globalnu vidljivost**.

Vanjski blok ne može pristupiti varijabli koja je definirana u unutarnjem bloku (lokalno definirane varijable). Međutim kada je varijabla deklarirana i dostupna vanjskom bloku (globalno definirane varijable), vidljiva je u svim ugniježđenim unutarnjim blokovima.

Primjer:

```
DECLARE
    -- Globalna varijabla
    broj1 NUMERIC := 7;
    broj2 NUMERIC := 21;
    broj3 NUMERIC := 36;
BEGIN
    DBMS_OUTPUT.PUT_LINE ('Globalna varijabla 1 - vanjski: ' || broj1);
    DBMS_OUTPUT.PUT_LINE ('Globalna varijabla 2 - vanjski: ' || broj2);
    DECLARE
        -- Lokalna varijabla
        broj1 NUMERIC := 55;
        broj2 NUMERIC := 47;
    BEGIN
        DBMS_OUTPUT.PUT_LINE ('Lokalna varijabla 1 - unutarnji: ' || broj1);
        DBMS_OUTPUT.PUT_LINE ('Lokalna varijabla 2 - unutarnji: ' || broj2);
        DBMS_OUTPUT.PUT_LINE ('Lokalna varijabla 3 - unutarnji: ' || broj3);
    END;
END;
/
```

Rezultat:

```
Globalna varijabla 1 - vanjski: 7
Globalna varijabla 2 - vanjski: 21
Lokalna varijabla 1 - unutarnji: 55
Lokalna varijabla 2 - unutarnji: 47
Lokalna varijabla 3 - unutarnji: 36

PL/SQL procedure successfully completed.
```

U primjeru, kreira se vanjski i unutarnji blok unutar kojega se definiraju varijable koje su globalno (vanjski blok) i lokalno (unutarnji blok) vidljive. Treba napomenuti da je u vanjskom bloku definirana varijabla *broj3* koja je vidljiva i unutarnjem bloku. Vrijednosti koje se ispisuju za vanjski blok su 7 i 21, dok se za unutarnji blok ispisuju vrijednosti 55 i 47 ali se i ispisuje vrijednost globalno definirane varijable, a to je vrijednost 36.

5.5 Usidreni tipovi podataka

Varijable se mogu deklarirati i dinamički te one ne nasljeđuju NOT NULL ograničenje stupca tablice, odnosno moguće je dinamičkoj varijabli pridružiti NULL vrijednost. Tip podataka koji se dodijeli varijabli može se temeljiti na objektu baze podataka. To se zove usidrena deklaracija zato što tip podataka varijable ovisi o objektu. Preporučljivo je da se usidreni tipovi podataka koriste kada je to god moguće, kako se ne bi morao mijenjati PL/SQL kod svaki put kada se mijenjaju tipovi podataka osnovnih objekata zato što se tipovi podataka uzimaju direktno iz tablica. PL/SQL nudi dva tipa „*sidrenja*“, a to su %TYPE i %ROWTYPE [18] [19].

5.5.1 %TYPE

Ako se varijabla koja se deklarira izravno preslikava na stupac tablice u bazi podataka, moguće je *usidriti* (eng. *Anchor*) tip varijable na tip podataka koji se nalazi u stupcu u bazi podataka pomoću atributa %TYPE. Varijable i konstante koje su deklarirane korištenjem atributa %TYPE promatraju se isto kao i one varijable i konstante koje su deklarirane korištenjem naziva podatkovnog tipa. Sintaksa za deklariranje varijable na temelju stupca je:

ime_varijable ime_tablice.ime_stupca%TYPE;

Ukoliko se promijeni struktura stupca, sve varijable koje su *usidrene* na tom stupcu, automatski će se promijeniti i vrijednost varijable do prvog sljedećeg korištenja programskog koda. Ako deklaracija nije bila *usidrena* i ako varijable nisu bile istog tipa na primjer DECIMAL(5,2) promjene na svim objektima trebale bi se napraviti pojedinačno.

5.5.2 %ROWTYPE

Atribut %ROWTYPE pridružuje varijabli tip zapisa (eng. *Record*) koji predstavlja redak u tablici ili pogledu. Polja u %ROWTYPE zapisu ne nasljeđuju ograničenja kao što su NOT NULL ili provjeru ograničenja ili zadane vrijednosti. Kada se promijeni struktura retka, mijenja se i struktura zapisa. Sintaksa za deklariranje varijable na temelju retka je:

ime_varijable ime_tablice_ili_ime_pogleda%ROWTYPE;

Primjer koda za model krojačnice:

```
DECLARE
    imeKrojac krojac.ime_krojaca%TYPE;
    prezimeKrojac krojac.prezime_krojaca%TYPE;
    nazivKrojacnice krojacnica.naziv_krojacnice%TYPE;
    redak krojac%ROWTYPE;
BEGIN
    SELECT k.ime_krojaca, k.prezime_krojaca, m.naziv_krojacnice
    INTO imeKrojac, prezimeKrojac, nazivKrojacnice
    FROM krojac k
    JOIN krojacnica m ON m.sifra_krojacnice = k.sifra_krojacnice
    WHERE k.sifra_krojaca = 32;
    DBMS_OUTPUT.PUT_LINE ('Krojač '
                                    || imeKrojac
                                    || ' '
                                    || prezimeKrojac
                                    || ' radi u krojačnici '
                                    || nazivKrojacnice);

    SELECT * INTO redak
    FROM krojac
    WHERE sifra_krojaca = 32;
    DBMS_OUTPUT.PUT_LINE( 'Plaća iznosi '|| redak.iznos_place || ' kuna' );
END;
/
```

Rezultat:

Krojač Melita Ožetski radi u krojačnici Stil
Plaća iznosi 3750 kuna

PL/SQL procedure successfully completed.

U primjeru definiraju se četiri varijable od kojih su tri deklarirane na temelju stupca (*ime*, *prezime*, *nazivKrojacnice*) dok je varijabla *redak* deklarirana na temelju retka. Pri deklariranju vidljivo je da se za pristupanje pojedinim poljima koristi točka (*krojac.ime_krojaca*). U prvom upitu ispisuje se ime i prezime krojača te naziv krojačnice u kojoj radi, dok u drugom upitu ispisuje se plaća krojača koja je dohvaćena atributom %ROWTYPE na cijeli redak.

5.6 Operatori

Operator manipulira pojedinačnim stavkama podataka te vraća rezultat, a te stavke podataka zovu se operandi ili argumenti. Za prikazivanje operatora koriste se posebni znakovi ili ključne riječi (na primjer za operator zbrajanja koristi se znak plusa (+)). Postoje dvije opće klase operatora, unarni (koristi samo jedan operand, format za unarni operator i njegov operand je oblika *operator operand*) i binarni (koristi dva operanda za rad, format za binarni operator i njegove operande je oblika *operand1 operator operand2*) [20] [21] [22]. PL/SQL nudi bogat izbor operatora, a oni se dijele na sljedeće tipove:

Tip operatora	Oznaka	Opis
Aritmetički operatori – koriste se za izvođenje različitih matematičkih operacija nad operandima	+	Operator zbrajanja omogućuje dodavanje prvog i drugog operanda; a + b
	-	Operator oduzimanja omogućuje da se prvi operand oduzme od drugoga operanda; a - b
	*	Operator množenja omogućuje da se prvi operand pomnoži s drugim operandom; a * b
	/	Operator dijeljenja omogućuje da se prvi operand podijeli s drugim operandom; a / b
	**	Operator potenciranja omogućuje da se prvi operand potencira na drugi; a ** b
	=	Operator jednakosti provjerava jesu li vrijednosti dva navedena operanda jednake ili ne. Ako je uvjet jednak onda će vratiti TRUE, inače FALSE; (a = b) je FALSE
	!= < > ~=	Operatori različitosti koriste se za provjeru jesu li dva operanda međusobno jednaka ili nemaju iste

Relacijski operatori – koriste se kada se uspoređuju dva izraza ili vrijednosti, a rezultat je BOOLEAN vrijednost		vrijednosti. Ako nisu jednaki onda će uvjet biti TRUE, inače FALSE; a != b je TRUE
	<	Usporedba vrijednosti manje od provjerava da li je vrijednost prvog operanda manja od vrijednosti drugog operanda, ako je, uvjet je TRUE*;* a < b je TRUE
	>	Usporedba vrijednosti veće od provjerava da li je vrijednost prvog operanda veća od vrijednosti drugog operanda, ako je, uvjet je TRUE; a > b je FALSE
	<=	Manje ili jednako provjerava da li je vrijednost prvog operanda manja ili jednaka vrijednosti drugog operanda, ako je, uvjet je TRUE*;* a <= b je TRUE
	>=	Veće ili jednako provjerava da li je vrijednost prvog operanda veća ili jednaka vrijednosti drugog operanda, ako je, uvjet je TRUE; a >= b je FALSE
Logički operatori – koriste se za spajanje više izraza ili definiranje izraza s dva	AND	Operator koji vraća vrijednost TRUE kada su prvi i drugi operand istinit, a vraća FALSE kada je prvi i drugi ili jedan od njih FALSE*;* a AND b je FALSE

operanda, vraćaju vrijednost TRUE ili FALSE ovisno o operandima i izrazima koji okružuju logičke operatore	OR	Operator koji vraća vrijednost TRUE kada su oba ili bilo koji od njih TRUE, inače vraća FALSE kada su oba operanda FALSE*;* a OR b je TRUE
	NOT	Logički NOT je operator koji mijenja logičko stanje operanda. Ako je uvjet TRUE, onda će ga logički NOT operator pretvoriti u FALSE; NOT a je FALSE
Operatori usporedbe – koriste se za usporedbu jedne vrijednosti s drugom. Dobivene vrijednosti mogu biti TRUE, FALSE ili NULL	LIKE	Operator koji uspoređuje vrijednosti znaka, niza ili CLOB-a te vraća vrijednost TRUE ako vrijednost odgovara uzorku ili FALSE ako ne odgovara uzorku. Znakovi koji se koriste u svrhu podudaranja su: % koji se koristi za podudaranje niza bilo kojih znakova te _ koji se koristi za podudaranje samo s jednim znakom; SELECT * FROM *krojac* WHERE *ime_krojaca* LIKE *'M%'*; Nakon izvršenja upita, dobije se prikaz krojača kojima ime počinje na slovo M (Melita, Marko), gdje se % koristi za podudaranje niza bilo kojeg znaka nakon slova M. SELECT * FROM *klijent* WHERE *ime_klijenta* LIKE *'_____a'*; Nakon izvršenja upita, dobije se prikaz klijenata čije se ime sastoji od 8 slova ali završava na slovo a (Katarina, Danijela)

	BETWEEN	Operator koji određuje nalazi li se vrijednost unutar zadanog intervala. Rezultat poprima vrijednost TRUE ako je dobivena vrijednost veća ili jednaka početnoj vrijednosti i manja ili jednaka krajnjoj vrijednosti; SELECT * FROM *krojac* WHERE *iznos_place* BETWEEN *3000* AND *4100*; Nakon izvršenja upita, dobije se prikaz krojača kojim je iznos plaće između 3000 i 4100 kuna
	IN	Operator koji provjerava da li bilo koja vrijednost pripada nekom skupu vrijednosti. Ako se vrijednost nalazi na danom skupu, rezultat je TRUE, inače je FALSE*;* SELECT * FROM *krojacnica* WHERE *postanski_broj* IN (10000,51000); Nakon izvršenja upita, dobije se prikaz svih krojačnica kojima poštanski broj odgovara zadanom skupu vrijednosti
	IS NULL	Operator koji vraća vrijednost TRUE ako je vrijednost operanda NULL, inače FALSE; SELECT * FROM *siva* WHERE *napomena* IS NULL; Nakon izvršenja upita, dobiju se informacije o proizvodima čije je polje napomena u tablici prazno

Tablica 3 Prikaz operatora

Treba uzeti u obzir da se prvo izvršavaju oni operatori koji imaju prednost nad ostalima, odnosno ako neki izraz ima više operatora, najprije će se evaluirati operatori višeg prioriteta. Operatore s jednakim prioritetom Oracle procjenjuje s lijeva na desno unutar naredbe. Tablica 4 prikazuje operatore od najvećeg prioriteta do najmanjeg.

Operator	**Operacija**
****	Potenciranje
+, - (unarni operatori)	Identitet, negacija, označavaju pozitivan ili negativan izraz
*, / (binarni operatori)	Množenje, dijeljenje
+, - (binarni operatori), ||	Zbrajanje, oduzimanje, povezivanje
=, !=, <, >, <=, >=, IS NULL, LIKE, BETWEEN, IN	Usporedba
NOT (Logički operator)	Logička negacija
AND (Logički operator)	Konjunkcija
OR (Logički operator)	Disjunkcija

Tablica 4 Prikaz prioriteta operatora

Da bi se nadjačao prioritet u izrazu, mogu se koristiti zagrade. Kod ugniježđenih zagrada, izrazi u unutarnjim zagradama prije će se izvršiti nego izrazi u vanjskim zagradama.

Od preostalih operatora bitno je spomenuti i operatore skupa (UNION, UNION ALL, INTERSECT, MINUS) koji kombiniraju skupove redaka koji vraćaju upiti u jedan rezultat. Upiti koji sadrže operatore skupa nazivaju se složenim upitima. Svi operatori imaju jednaku razinu prioriteta.

6. Struktura PL/SQL programa

PL/SQL sadrži tri kategorije kontrolnih izraza:

- Uvjetni izrazi (eng. *Conditional selection statements*)
- Iterativni izrazi (eng. *Loop statements*)
- Sekvencijalni izrazi (eng. *Sequential control statements*)

Kontrolni izrazi najvažniji su unutar PL/SQL-a zato što oni kontroliraju tijek izvršavanja programa. PL/SQL pruža različite vrste naredbi za pružanje takve vrste proceduralnih mogućnosti. Testiranje uvjeta i petlji te mogućnost grananja dostupno je unutar PL/SQL-a što omogućuje pisanje dobro strukturiranih programa.

6.1 Uvjetni izrazi

Uvjetni izrazi ovise o tome koji blok naredbi treba izvršiti ili ne, o čemu odlučuje uvjet (eng. *Condition*). Ako je uvjet TRUE blok naredbi će se izvršiti, a ako je uvjet FALSE tada se blok naredbi neće izvršiti. Uvjetni izrazi pokreću različite naredbe za različite vrijednosti podataka. Dijele se na [1] [9] [23] [24] [25]:

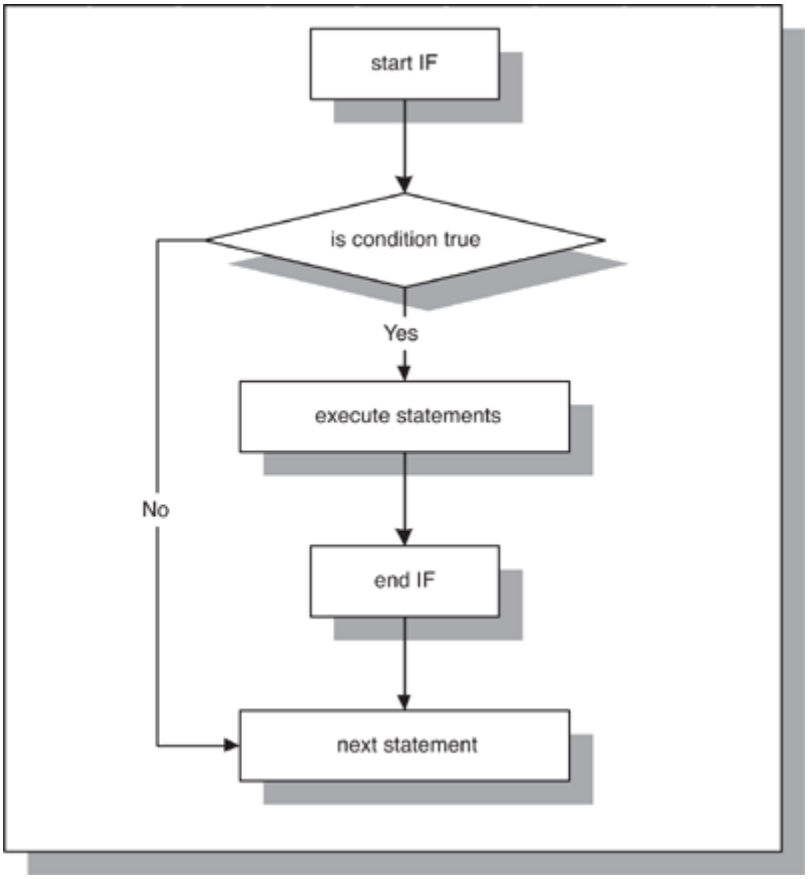
- IF naredbe
- CASE naredbe

IF-THEN: Naredba IF-THEN najosnovniji je oblik IF naredbe. Koristi se kada se provjerava jedan uvjet. Ako je rezultat uvjeta TRUE, izvršiti će se naredbe koje su između izraza THEN i END IF, a ako je rezultat uvjeta FALSE ili NULL, naredba se neće izvršiti nego će program jednostavno izaći iz bloka naredbi. Sintaksa za IF-THEN naredbu je:

```
IF  uvjet THEN
        naredbe 1;
        ...
        naredbe N;
END IF;
```

Rezervirana riječ IF označava početak IF naredbe. Naredbe od 1 do N su slijed izvršnih naredbi koje sadrže jednu ili više standardnih programskih struktura. Između ključnih riječi IF

i THEN nalazi je *uvjet* koji odlučuje hoću li se naredbe izvršiti. *uvjet* može biti bilo koji izraz, varijabla, konstanta ili identifikator. Rezervirana riječ END IF označava kraj IF-THEN naredbe.



Slika 6 Prikaz dijagrama toka za IF-THEN naredbu [1]

Kada se izvrši IF-THEN naredba, procjenjuje se da li je rezultat uvjeta TRUE ili FALSE. Ako je uvjet TRUE izvršava se sljedeća naredba nakon ključne riječi THEN. Ako je uvjet FALSE, program će izvršiti sljedeću naredbu koja dolazi nakon naredbe END IF.

Primjer koda za model krojačnice:

```
DECLARE iznos_place_krojaca NUMBER := 5000;
BEGIN
        IF iznos_place_krojaca > 3750 THEN
          DBMS_OUTPUT.PUT_LINE( 'Krojač ima plaću koja je veća od minimalne
plaće!!' );
        END IF;
END;
```

Rezultat:

Krojač ima plaću koja je veća od minimalne plaće!!

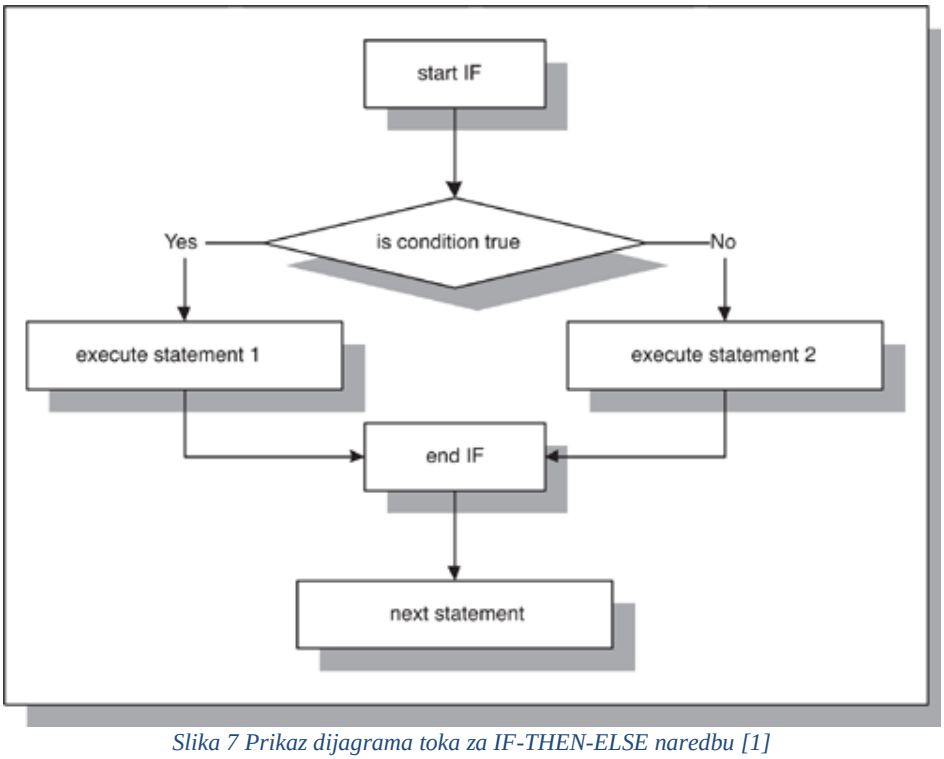
PL/SQL procedure successfully completed.

U primjeru vrijednost varijable *iznos_place_krojaca* postavljena je na 5 000. Uvjet se provjerava te je rezultat TRUE, odnosno izvršava se naredba između ključnih riječi THEN i END IF.

IF-THEN-ELSE: Naredbi IF dodaje se ključna riječ ELSE iza koje slijede naredbe. Omogućuje da se navedu dvije naredbe, ovisno o uvjetu tako da kada je uvjet TRUE izvršava se jedan skup naredbi, a ako je uvjet FALSE izvršava se drugi skup naredbi. Sintaksa za IF-THEN-ELSE naredbu je:

IF *uvjet* THEN
    n*aredbe 1;*
ELSE
    naredbe 2;
END IF;

Ako je *uvjet* TRUE, izvršavaju se *naredbe 1* između ključnih riječi THEN i ELSE, a ako je *uvjet* FALSE ili NULL izvršavaju se *naredbe 2* između ključnih riječi ELSE i END IF. Navedenu logiku izvođenja prikazuje slika 7.



Slika 7 Prikaz dijagrama toka za IF-THEN-ELSE naredbu [1]

Primjer koda za model krojačnice:

```
DECLARE
        cijena_sivanja NUMBER := 1200;
        iznos_popusta NUMBER (10,2) := 0;
BEGIN
        IF cijena_sivanja > 600 THEN
            iznos_popusta := cijena_sivanja * 0.15;
                DBMS_OUTPUT.PUT_LINE ('Cijena šivanja smanjenja je za ' ||
iznos_popusta || ' kuna.');
        ELSE
            iznos_popusta := cijena_sivanja * 0.00;
                DBMS_OUTPUT.PUT_LINE ('Cijena šivanja smanjenja je za ' ||
iznos_popusta || ' kuna.');
        END IF;
END;
```

Rezultat:

Cijena šivanja smanjenja je za 180 kuna.

PL/SQL procedure successfully completed.

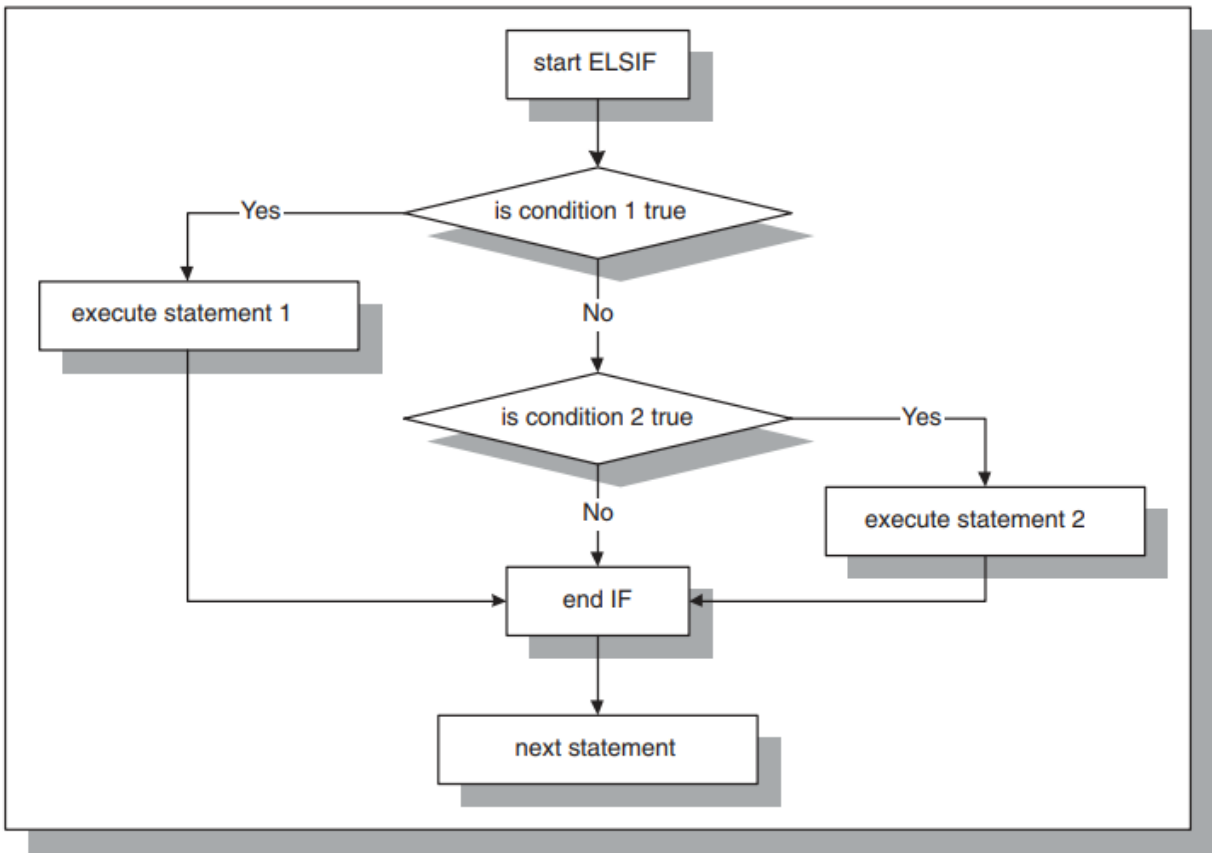
U primjeru definirale su se dvije varijable *cijena_sivanja* i *iznos_popusta.* Uvjet se provjerava i rezultat uvjeta je TRUE te se izvršava prva naredba.

IF-THEN-ELSIF: Naredba IF-THEN-ELSIF koristi se za provjeru više uvjeta u kojoj se svi uvjeti testiraju jedan po jedan. Ako je *uvjet* TRUE taj blok naredbi se izvršava, a ako je FALSE izvršava se drugi blok naredbi. Sintaksa za IF-THEN-ELSIF naredbu je:

```
IF uvjet 1 THEN
        naredbe 1;
ELSIF uvjet 2 THEN
        naredbe 2;
ELSIF uvjet 3 THEN
        naredbe 3;
…
ELSE
        naredbe N;
END IF;
```

Rezervirana riječ IF označava početak ELSIF konstrukcije. *uvjet 1* do *uvjet N* je skup uvjeta za koje se provjerava da li su TRUE ili FALSE. Ako je *uvjet 1* koji se nalazi između ključnih

riječi IF i THEN istinit, odnosno TRUE (ostali se uvjeti onda ne vrednuju) *naredba 1* se izvršava i kontrola se daje prvoj izvršnoj naredbi nakon rezervirane riječi END IF. No u slučaju da je *uvjet 1* FALSE, kontrola se prosljeđuje na ELSIF dio i *uvjet 2* se provjerava da li TRUE ili FALSE i tako dalje za sve ostale uvjete sve dok jedan ne bude TRUE. Ako se ispostavi da niti jedan uvjet nije TRUE izvršavaju se *naredbe N* između ključnih riječi ELSE i END IF. U slučaju da se preskoči ELSE klauzula i da niti jedan uvjet nije TRUE, tada IF-THEN-ELSIF naredba ne bi izvršila ništa. Slika 8 prikazuje logiku izvođenja za IF-THEN-ELSIF naredbu.



Slika 8 Prikaz dijagrama toka za IF-THEN-ELSIF naredbu [1]

Potrebno je napomenuti da se uvjetni izrazi mogu ugnijezditi jedan unutar drugoga. Naredba IF, na primjer može biti ugniježđena unutar naredbe ELSIF i obrnuto. Međutim ako kod sadrži previše ugniježđenih IF naredbi, biti će ga teško čitati i održavati te bi stoga trebalo izbjegavati ugniježđene IF naredbe.

Primjer koda za model krojačnice:

```
DECLARE
        iznos_popusta  NUMBER(6,2) := 0;
```

```
        cijena_sivanja NUMBER(6) := 400;
    BEGIN
        IF cijena_sivanja > 600 THEN
        iznos_popusta := cijena_sivanja * 0.15;
    ELSIF cijena_sivanja BETWEEN 300 AND 599 THEN
        iznos_popusta :=  cijena_sivanja * 0.05;
    ELSE
        iznos_popusta := cijena_sivanja * 0.0;
    END IF;
    DBMS_OUTPUT.PUT_LINE ('Cijena šivanja smanjena je za ' || iznos_popusta || '
kuna.');
    END;
    /
```

Rezultat:

Cijena šivanja smanjena je za 20 kuna.

PL/SQL procedure successfully completed.

Kod je isti kao i za prethodni primjer samo je dodano više uvjeta. Ponovno se provjeravaju uvjeti jedan po jedan. U danom primjeru drugi uvjet je TRUE te se on izvršava.

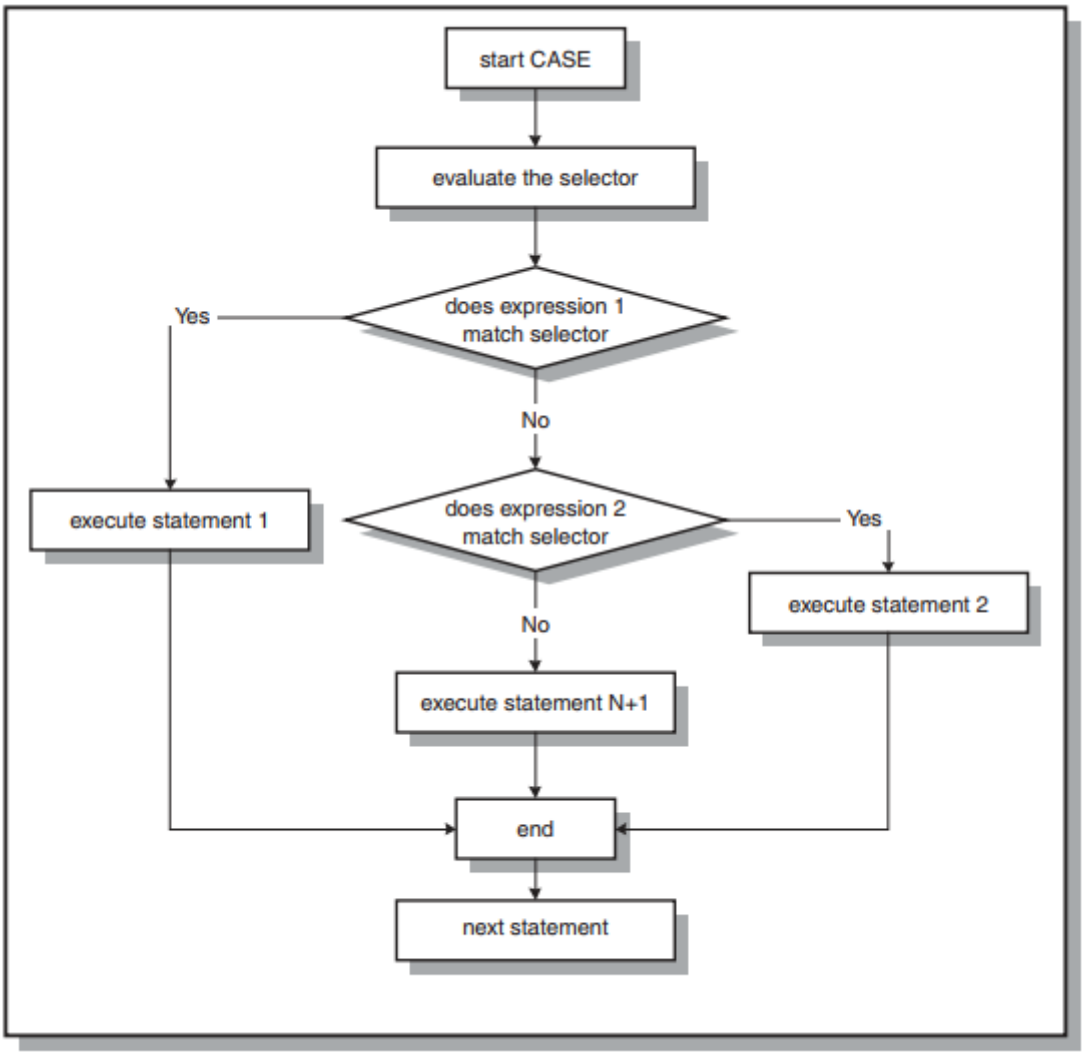
CASE: Postoje dvije vrste CASE naredbi, jednostavne CASE naredbe (eng. *Simple CASE Statement*) i tražene CASE naredbe (eng. *Searched CASE Statement*). Naredba CASE, kao i naredba IF odabire jedan slijed naredbi od više naredbi koje će se izvršiti s naglaskom da CASE naredba koristi selektor[12] umjesto više uvjeta kojima se provjerava istinitost dok tražena CASE naredba ne koristi selektor nego koristi uvjete koji se provjeravaju korištenjem WHEN klauzule. Ako je uvjet TRUE naredbe se izvršavaju, a ako je FALSE uvjeti se provjeravaju uzastopno pomoću WHEN klauzule. Ako se selektor ne navede, PL/SQL dodaje TRUE kao selektor. Za razliku od IF-THEN-ELSIF, naredba CASE može se koristiti i u SQL izrazima. Sintaksa za CASE naredbu je:

```
CASE selektor
    WHEN uvjet 1 THEN naredbe 1;
    WHEN uvjet 2 THEN naredbe 2;
    …
    WHEN uvjet N THEN naredbe N;
    ELSE naredbe N+1;
```

[12] Selektor je varijabla, funkcija ili izraz (osim BLOB-a, BFILE-a ili kompozitnih tipova podataka) koje CASE naredba pokušava ispitati u WHEN bloku.

END CASE;

Rezervirana riječ CASE označava početak CASE naredbe. Selektor je vrijednost koja određuje koja se WHEN klauzula treba izvršiti. Selektor se navodi samo jednom, dok WHEN klauzule uzastopno. Svaka se WHEN klauzula sastoji od *uvjeta* te jedne ili više naredbi koje se uspoređuju s vrijednošću selektora. Ako su vrijednosti jednake, izvršava se ona naredba koja je povezana s određenom WHEN klauzulom, dok se preostale zanemaruju. Ako niti jedna naredba ne odgovara vrijednosti selektora, izvršava se ELSE klauzula koja nije obavezna (funkcionira slično kao i u IF-THEN-ELSE naredbi). Rezervirana riječ END CASE označava kraj CASE naredbe. Slika 9 prikazuje logiku izvođenja CASE naredbe.



Slika 9 Prikaz dijagrama toka za CASE naredbu [1]

Primjer koda za model krojačnice:

```
DECLARE
   postanski_broj NUMBER;
BEGIN
   postanski_broj := 10310;
  CASE postanski_broj
     WHEN 51000 THEN DBMS_OUTPUT.PUT_LINE ('Grad u kojem se krojačnica
nalazi je Rijeka.');
     WHEN 10000 THEN DBMS_OUTPUT.PUT_LINE ('Grad u kojem se krojačnica
nalazi je Zagreb.');
     WHEN 31000 THEN DBMS_OUTPUT.PUT_LINE ('Grad u kojem se krojačnica
nalazi je Osijek.');
     WHEN 10310 THEN DBMS_OUTPUT.PUT_LINE ('Grad u kojem se krojačnica
nalazi je Ivanić Grad.');
     WHEN 21000 THEN DBMS_OUTPUT.PUT_LINE ('Grad u kojem se krojačnica
nalazi je Split.');
     ELSE DBMS_OUTPUT.PUT_LINE ('Niti jedan poštanski broj ne odgovara
nazivu mjesta!!');
END CASE;
END;
```

Rezultat:

Grad u kojem se krojačnica nalazi je Ivanić Grad.

PL/SQL procedure successfully completed.

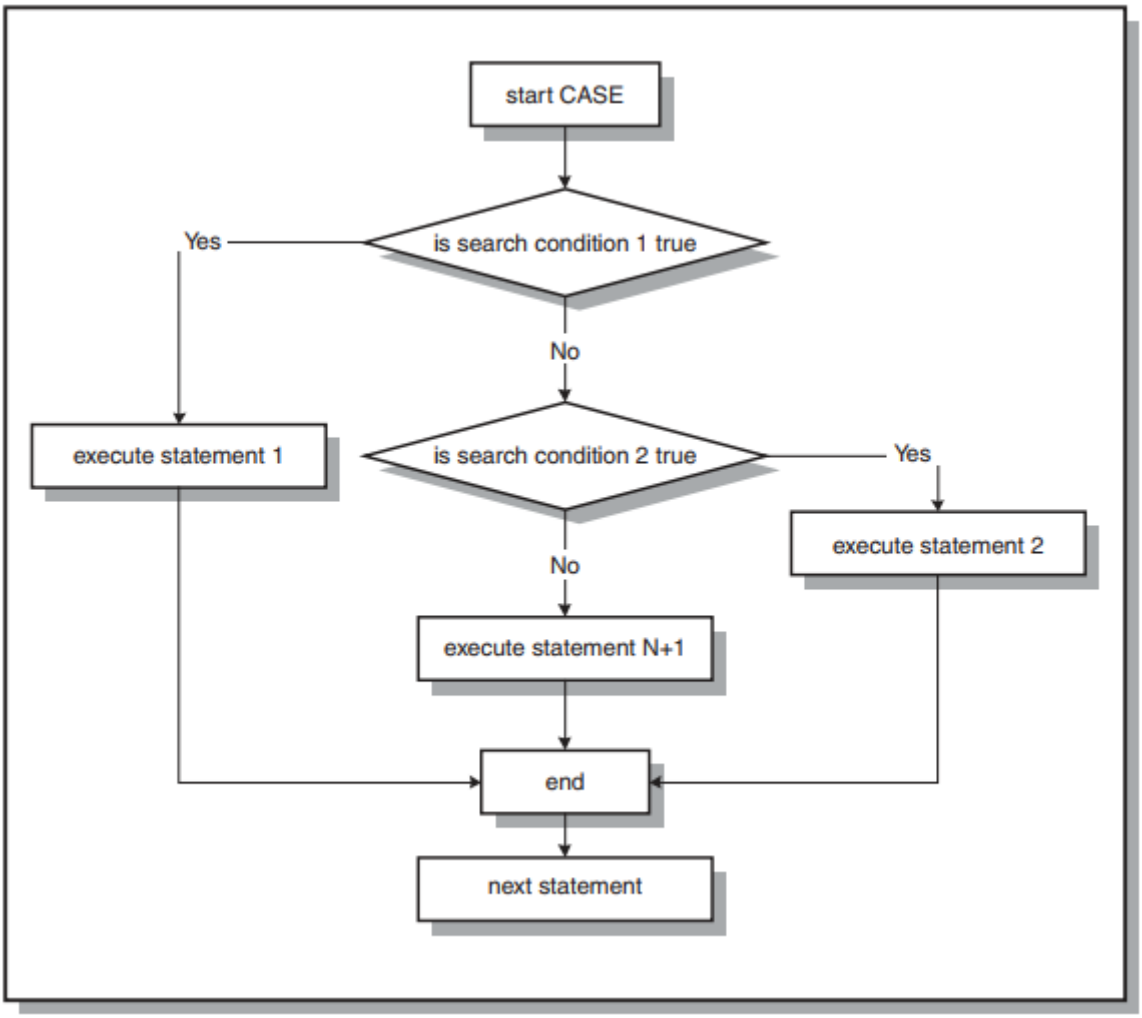
U primjeru za prikaz jednostavne CASE naredbe koristio se poštanski broj koji je trebao odgovarati točnom mjestu, odnosno koristilo se pet Boolean izraza. U svakom slučaju testiralo se da li je varijabla *postanski_broj* jednaka jednoj od pet navedenih vrijednosti.

Sintaksa za traženu CASE naredbu:

```
CASE
     WHEN  traženi uvjet 1 THEN naredbe 1;
     WHEN traženi uvjet 2 THEN naredbe 2;
     …
     WHEN traženi uvjet N THEN naredbe N;
     ELSE naredbe N+1;

END CASE;
```

Tražena CASE naredba ima *tražene uvjete* koji vraćaju TRUE, FALSE ili NULL. Kada je određeni traženi uvjet TRUE izvršava se grupa naredbi koja je povezana s tim uvjetom. Isto kao kod jednostavne CASE naredbe, ako niti jedan *traženi uvjet* nije istinit odnosno TRUE, izvršavaju se naredbe koje su povezane s ELSE klauzulom koja također nije obavezna. Slika 10 prikazuje logiku izvođenja tražene CASE naredbe.



Slika 10 Prikaz dijagrama toka za traženu CASE naredbu [1]

Primjer koda za model krojačnice:

```
DECLARE
   postanski_broj NUMBER;
BEGIN
   postanski_broj := 10310;
   CASE
     WHEN postanski_broj = 51000 THEN DBMS_OUTPUT.PUT_LINE('Grad u
kojem se krojačnica nalazi je Rijeka.');
```

```
        WHEN postanski_broj = 10000 THEN DBMS_OUTPUT.PUT_LINE('Grad u
kojem se krojačnica nalazi je Zagreb.');
        WHEN postanski_broj = 31000 THEN DBMS_OUTPUT.PUT_LINE('Grad u
kojem se krojačnica nalazi je Osijek.');
        WHEN postanski_broj = 10310 THEN DBMS_OUTPUT.PUT_LINE('Grad u
kojem se krojačnica nalazi je Ivanić Grad.');
        WHEN postanski_broj = 21000 THEN DBMS_OUTPUT.PUT_LINE('Grad u
kojem se krojačnica nalazi je Split.');
        ELSE DBMS_OUTPUT.PUT_LINE('Niti jedan poštanski broj ne odgovara nazivu
mjesta!!');
      END CASE;
    END;
```

Rezultat:

Grad u kojem se krojačnica nalazi je Ivanić Grad.

PL/SQL procedure successfully completed.

U varijablu *postanski_broj* spremljena je vrijednost poštanskog broja za mjesto Ivanić Grad. Provjerom Boolean vrijednosti svakog uvjeta određuje se koja će se WHEN klauzula izvršiti, odnosno koja je vrijednost varijable *postanski_broj* jednaka jednoj od navedenih vrijednosti.

6.2 Iterativni izrazi

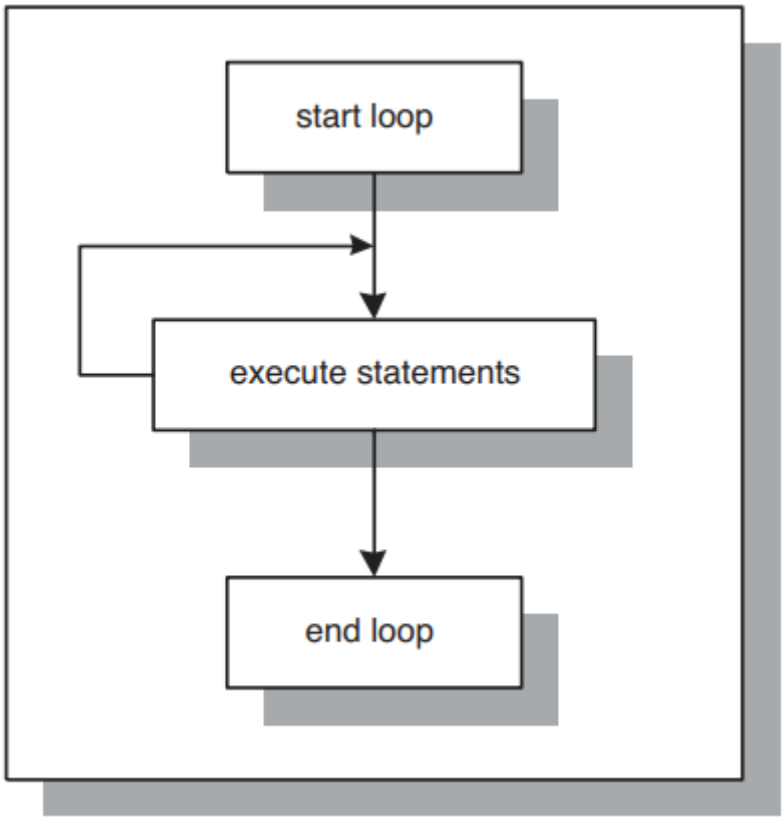
Iterativni izrazi su blokovi koji omogućuju ponavljanje jedne naredbe ili skupa naredbi sve dok uvjet nije ispunjen. Postoje tri vrste petlji unutar PL/SQL-a:

- Jednostavna petlja
- FOR petlja
- WHILE petlja

Jednostavna petlja: Jednostavna petlja je najosnovnija vrsta petlje. Sintaksa za jednostavnu petlju je:

```
LOOP
        naredbe 1;
        naredbe 2;
        …
        naredbe N;
END LOOP;
```

Jednostavna petlja započinje ključnom riječju LOOP. Naredbe od 1 do N su niz naredbi koje se ponavljaju sve dok se petlja ne zaustavi. Kraj petlje označava se ključnom riječju END LOOP. Slika 11 prikazuje logiku izvođenja jednostavne petlje.



Slika 11 Prikaz dijagrama roka za jednostavnu petlju [1]

Jednostavna petlja izvodi se sve dok se eksplicitno ne zaustavi korištenjem naredbi EXIT (odmah se izlazi iz petlje kada je uvjet istinit odnosno TRUE te se kontrola toka programa prosljeđuje na prvu naredbu nakon ključne riječi END LOOP) ili EXIT WHEN (iz petlje se izlazi nakon što se ostvari određeni uvjet). Za provjeru uvjeta, naredba EXIT koristi IF naredbe. Sintaksa za EXIT naredbu je:

```
LOOP
        naredbe 1;
        naredbe 2;
        IF uvjet THEN
                EXIT;
        END IF;
END LOOP:
naredbe 3;
```

Sintaksa za EXIT WHEN naredbu je:

```
LOOP
        naredbe 1;
        naredbe 2;
        EXIT WHEN uvjet;
END LOOP;
naredbe 3;
```

Kod EXIT WHEN naredbe, tijekom svake iteracije petlja izvršava niz naredbi. Kontrola toka programa prosljeđuje se *uvjetu* EXIT WHEN naredbe. Ako je vrijednost uvjeta FALSE, petlja se ponovno izvršava sve dok uvjet ne bude TRUE te se nakon toga petlja prekida. Nakon toga kontrola toka programa prosljeđuje se prvoj naredbi koja slijedi nakon petlje. Nakon izlaska iz petlje prikazana je vrijednost varijable *brojac.*

Primjer:

```
DECLARE
        brojac NUMBER := 0;
BEGIN
        LOOP
                brojac := brojac + 1;
                IF brojac > 5 THEN
                        EXIT;
                END IF;
                DBMS_OUTPUT.PUT_LINE( 'Unutarnja petlja: ' || brojac )  ;
        END LOOP;
        -- kontrola se nastavlja nakon EXIT naredbe
        DBMS_OUTPUT.PUT_LINE( 'Vrijednost nakon izlaska iz petlje: ' || brojac );
END;
```

Rezultat:

Unutarnja petlja: 1
Unutarnja petlja: 2
Unutarnja petlja: 3
Unutarnja petlja: 4
Unutarnja petlja: 5
Vrijednost nakon izlaska iz petlje: 6

PL/SQL procedure successfully completed.

Za primjer jednostavne petlje s EXIT naredbom deklarirala se i inicijalizirala varijabla *brojac* s početnom vrijednošću postavljenu na 0. Vrijednost brojača povećava se za jedan nakon

svake iteracije unutar petlje i iz petlje se izlazi ako je vrijednost brojača veća od 5 (kako je zadano u IF petlji). Ako je vrijednost brojača jednaka 5 pokaži vrijednost varijable *brojac.* Kako je na početku vrijednost varijable *brojac* postavljena na 0 kod unutar tijela petlje izvodi se pet puta prije nego se prekine.

Primjer:

```
DECLARE
    brojac NUMBER := 0;
BEGIN
    LOOP
        DBMS_OUTPUT.PUT_LINE('Sveučilište u Rijeci');
        brojac := brojac + 1;
        EXIT WHEN brojac = 7;
    END LOOP;
END;
/
```

Rezultat:

Sveučilište u Rijeci
Sveučilište u Rijeci
Sveučilište u Rijeci
Sveučilište u Rijeci
Sveučilište u Rijeci
Sveučilište u Rijeci
Sveučilište u Rijeci

PL/SQL procedure successfully completed.

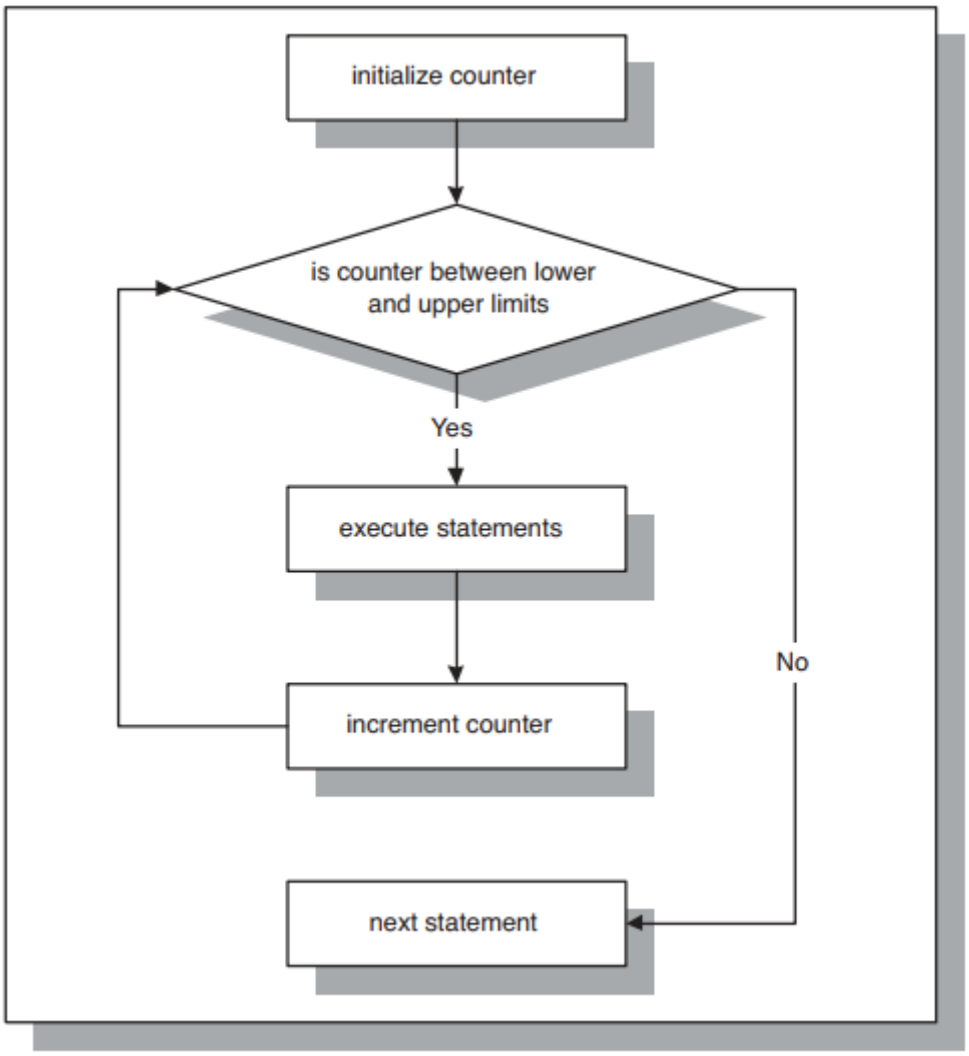
Za primjer jednostavne petlje s EXIT WHEN naredbom deklarirala i inicijalizirala se varijabla *brojac* s početnom vrijednošću 0. Petlja izvodi sedam iteracija i izlazi se iz petlje kada je uvjet zadovoljen.

FOR petlja: For petlja koristi se kada se niz naredbi treba izvršiti određeni broj puta. Sintaksa za FOR petlju je:

```
FOR brojac IN [REVERSE] donja_granica..gornja_granica LOOP
    naredbe 1;
    naredbe 2;
    ...
    naredbe N;
```

END LOOP;

Rezervirana riječ FOR označava početak FOR petlje. Potrebno je deklarirati varijablu *brojac* koja odlučuje koliko puta će se petlja izvršiti na temelju donje i gornje vrijednosti koje su navedene na početku petlje. *donja_granica..gornja_granica* definira broj iteracija petlje i njihova vrijednost provjerava se jednom kod prve iteracije. Naredbe od 1 do N su niz naredbi koje se više puta ponavljaju. Rezervirana riječ END LOOP označava kraj FOR petlje. Rezervirana riječ IN ili IN REVERSE sastavni je dio petlje. Korištenjem rezervirane riječi REVERSE broj ponavljanja petlje počinje od *gornja_granica* i smanjuje se za jedan sa svakom iteracijom petlje sve dok ne dosegne *donja_granica*. Slika 12 prikazuje logiku izvođenja FOR petlje.



Slika 12 Prikaz dijagrama toka za FOR petlju [1]

brojac petlje inicijaliziran je na *donja_granica* kod prve iteracije. Vrijednost brojača petlje provjerava se za svaku iteraciju. Petlja se izvodi samo u slučaju kada vrijednost brojača kreće

od *donja_granica* prema *gornja_granica*. Ukoliko vrijednost brojača ne zadovoljava raspon koji je određen donjom i gornjom granicom, kontrola toka programa prosljeđuje se prvoj naredbi koja se nalazi izvan petlje.

Primjer:

```
DECLARE

    brojac NUMBER(2);
BEGIN
    FOR brojac IN 1..7 LOOP
        DBMS_OUTPUT.PUT_LINE(brojac);
    END LOOP;
END;
/
```

Rezultat:

1
2
3
4
5
6
7

PL/SQL procedure successfully completed.

Rezultat FOR petlje pokazuje da nakon svake iteracije petlje, broj se povećava za jedan sve dok se petlja ne izvrši do kraja.

Primjer FOR petlje koja sadrži rezerviranu riječ IN REVERSE.

```
DECLARE
    brojac NUMBER := 1;
BEGIN
    FOR i IN REVERSE brojac..5 LOOP
        DBMS_OUTPUT.PUT_LINE('Brojač petlje: ' || i);

    END LOOP;
END;
/
```

Rezultat:

Brojač petlje: 5

Brojač petlje: 4
Brojač petlje: 3
Brojač petlje: 2
Brojač petlje: 1

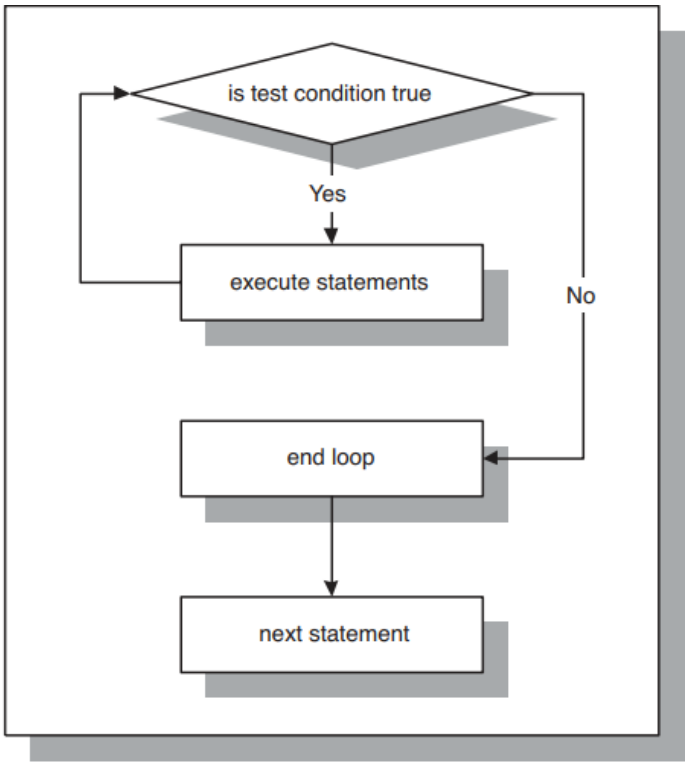
PL/SQL procedure successfully completed.

Početna vrijednost varijable *brojac* postavljena je na 1, odnosno do koje vrijednosti će se petlja izvršavati. Nakon svake iteracije vrijednost varijable smanjuje se za jedan sve dok petlja ne dosegne u ovom slučaju vrijednost jedan.

WHILE petlja: WHILE petlja se izvršava sve dok je vrijednost uvjeta TRUE. Sintaksa za WHILE petlju je:

```
WHILE uvjet LOOP
        naredbe 1;
        naredbe 2;
        ...
        naredbe N;
END LOOP;
```

Rezervirana riječ WHILE označava početak WHILE petlje. Prilikom provjere *uvjet* može biti TRUE, FALSE ili NULL. Ako je TRUE petlja se izvršava, a kada je FALSE ili NULL petlja se prekida i kontrola toka programa prosljeđuje se sljedećoj naredbi nakon petlje. Naredbe od 1 do N su naredbe koje se izvršavaju više puta. Rezervirana riječ END LOOP označava kraj petlje. Slika 13 prikazuje logiku izvođenja WHILE petlje.



Slika 13 Prikaz dijagrama toka za WHILE petlju [1]

PL/SQL provjerava *uvjet* u WHILE klauzuli prije svake iteracije petlje. Ako je vrijednost uvjeta FALSE prije ulaska u petlju, WHILE petlja neće se izvršiti. To je razlika u odnosu na jednostavne petlje čije se tijelo petlje uvijek izvrši jednom. Ključno je osigurati da u nekom trenutku vrijednost uvjeta poprimi vrijednost FALSE zato što će se petlja izvršavati neprekidno. Za prekid petlje koriste se naredbe EXIT i EXIT WHEN.

Primjer:

```
DECLARE
        brojac NUMBER := 1;
BEGIN
        WHILE brojac <= 7
                LOOP
                        DBMS_OUTPUT.PUT_LINE( 'Vrijednost brojača : ' || brojac );
                        brojac := brojac + 1;
                        EXIT WHEN brojac = 4;
                END LOOP;
        END;
        /
```

Rezultat:

Vrijednost brojača : 1
Vrijednost brojača : 2
Vrijednost brojača : 3

PL/SQL procedure successfully completed.

Za primjer WHILE petlje deklarira se i inicijalizira varijabla *brojac* i vrijednost se postavi na jedan. Uvjet unutar WHILE klauzule provjerava se prije svake iteracije petlje. Unutar tijela petlje svakom iteracijom *brojac* se povećava za jedan. Uvjet unutar EXIT WHEN klauzule ima vrijednost TRUE kada *brojac* poprimi vrijednost četiri. To je razloga zašto se je tijelo petlje izvršilo samo tri puta prije završetka petlje.

6.3 Sekvencijalni izrazi

Sekvencijalni izrazi nisu presudni za PL/SQL programiranje. GOTO naredba određuje pomak u izvršavanju koda na definirani dio (oznaku). GOTO naredbe moraju se koristiti umjereno iz razloga što pretjerano korištenje uzrokuje teško razumijevanje i održavanje koda [1] [26] [27]. Sintaksa za GOTO naredbu je:

GOTO *naziv_oznake;*

naziv_oznake je naziv oznake koja prepoznaje ciljanu naredbu te je okružena dvostrukim izlomljenim zagradama:

<<*naziv_oznake*>>

Dvostruke izlomljene zagrade znakovi su za odvajanje, a *naziv_oznake* ponaša se kao pokazivač koji je dostupan tijekom izvođenja programa. Postoji nekoliko pravila za GOTO naredbe:

- GOTO naredba ne može se koristiti za prijenos kontrole toka programa u IF, CASE ili LOOP naredbi te u podbloku
- GOTO naredba ne može se koristiti za prijenos kontrole toka programa s jedne klauzule na drugu u IF naredbama (na primjer iz IF klauzule na ELSIF ili ELSE klauzulu ili s WHEN klauzule na CASE naredbu)
- GOTO naredba ne može se koristit za prijenos kontrole toka programa iz potprograma ili u dio za obradu iznimaka

- GOTO naredba ne može se koristiti za prijenos kontrole toka programa iz dijela za obradu iznimaka natrag u trenutni blok

Primjer koda za model krojačnice:

```
DECLARE
        var_ime_krojaca VARCHAR (30);
        var_sifra_krojaca NUMBER(2) := 96 ;
BEGIN
        <<get_ime>>
        SELECT ime_krojaca INTO var_ime_krojaca
        FROM krojac
        WHERE sifra_krojaca = var_sifra_krojaca;
BEGIN
        DBMS_OUTPUT.PUT_LINE(var_ime_krojaca);
        var_sifra_krojaca := var_sifra_krojaca + 1;
        IF var_sifra_krojaca < 96 THEN
     GOTO get_ime;
   END IF;
  END;
END;
/
```

Rezultat:

Mirjana

PL/SQL procedure successfully completed.

7. Tipovi podataka (2/2)

U poglavlju 5.3 opisani su skalarni tipovi podataka i veliki objekti. U ovom poglavlju detaljnije su obrađeni referencirani i kompozitni tipovi podataka.

7.1 Referencirani tip podataka (eng. *Reference Data Type*)

Glavna razlika između referenciranih i drugih tipova podataka je način na koji upravljaju memorijom i pohranom. Referencirani tip podataka pruža memorijske strukture, ali za razliku od skalarnih i kompozitnih tipova podataka mogu pokazivati na različita mjesta pohrane. Za izvršavanje SQL naredbi i pohranjivanje određenih informacija Oracle koristi radna područja. Referenciranom tipu podataka pripadaju kursori (eng. *Cursor*) koji omogućuju imenovanje radnog područja i pristupanje informacijama pohranjenim u tom području [28]. Kursor sadrži jedan ili više redaka koje vraća SQL naredba. Skup redaka koji su pohranjeni unutar kursora nazivaju se aktivnim skupom podataka (eng. A*ctive data set*). Kursori se dijele na:

- Implicitne kursore (eng. *Implicit cursors*)
- Eksplicitne kursore (eng. *Explicit cursors*)

7.1.1 Implicitni kursori

Kursor koji je automatski kreiran od strane Oraclea dok izvršava bilo koje DML naredbe kao što su INSERT, UPDATE i DELETE naziva se implicitni kursor. Budući da korisnik toga nije svjestan, ne može kontrolirati ili obraditi informacije u implicitnom kursoru. Isto vrijedi i za programere. Za naredbu INSERT kursor sadrži podatke koji se trebaju spremiti u bazu podataka. Za naredbe UPDATE i DELETE kursor prepoznaje retke koji će se obrisati ili ažurirati [29]. Sintaksa atributa SQL kursora glasi SQL*atribut.* Najčešće korišteni atributi su %FOUND, %ISOPEN, %NOTFOUND i %ROWCOUNT. U slučaju da se niti jedna naredba ne izvrši, vrijednost SQL*atribut* je NULL. Za eksplicitni kursor umjesto SQL naziva dodaje se ime kursora, odnosno sintaksa glasi *ime_kursora%atribut.* Tablica 5 prikazuje detaljniji opis atributa koje kursori koriste [30].

Naziv atributa	Opis
%FOUND	U slučaju kada će naredbe INSERT, UPDATE ili DELETE obrisati ili ažurirati jedan ili više redaka ili ako je naredba SELECT INTO vratila jedan ili više redaka, funkcija %FOUND vraća TRUE. U suprotnome vraća FALSE
%NOTFOUND	U slučaju kada naredbe INSERT, UPDATE ili DELETE neće obrisati ili ažurirati retke ili ako naredba SELECT INTO neće vratiti niti jedan redak, naredba %NOTFOUND vraća TRUE. U suprotnome vraća FALSE.
%ISOPEN	U slučaju da je kursor otvoren, vraća se vrijednost TRUE, u suprotnome FALSE. Budući da Oracle automatski zatvara SQL kursor nakon izvršenja pridružene SQL naredbe, uvijek vraća FALSE za implicitne kursore
%ROWCOUNT	Vraća broj redaka kada naredbe INSERT, UPDATE ili DELETE obrišu ili ažuriraju retke ili se broj redaka vrati korištenjem SELECT INTO naredbe

Tablica 5 Prikaz atributa SQL kursora

U slučaju kada upit vraća nulu ili više redaka, implicitni kursor uzrokuje probleme, to jest vraća NO_DATA_FOUND odnosno iznimku TOO_MANY_ROWS.

Primjer koda za model krojačnice:

```
DECLARE
        var_sifra_krojacnice NUMBER := &sifra_krojacnice;
BEGIN
        UPDATE krojacnica
        SET adresa = 'Matka Baštijana 55'
        WHERE sifra_krojacnice = var_sifra_krojacnice;
 IF SQL%NOTFOUND THEN
```

```
        DBMS_OUTPUT.PUT_LINE ('Ne postoji krojačnica pod šifrom '||
var_sifra_krojacnice||'. Promijeni unos!!');
  ELSE
        DBMS_OUTPUT.PUT_LINE('Adresa je uspješno ažurirana za krojačnicu čija
je šifra: ' || var_sifra_krojacnice);
  END IF;
END;
/
```

Rezultat:

	SIFRA_KROJACNICE	NAZIV_KROJACNICE	ADRESA	POSTANSKI_BROJ
1	32	Igla	Savska 5	10000
2	214	Botun	Vukovarska 8	51000
3	7	Stil	Dolanec 37	10310
4	1	Šivaona	Opatinec 4	10310
5	3	Jenny	Ulica Ruža 1	51000
6	4	Bolero	Ulica kneza Branimira 19	10000
7	5	Kreacija	Južna ulica 55	51000
8	6	As	Dubečka ulica 21	10000

Slika 14 Prikaz naziva krojačnica čija je šifra 32 prije ažuriranja naziva

	SIFRA_KROJACNICE	NAZIV_KROJACNICE	ADRESA	POSTANSKI_BROJ
1	32	Igla	Matka Baštijana 55	10000
2	214	Botun	Vukovarska 8	51000
3	7	Stil	Dolanec 37	10310
4	1	Šivaona	Opatinec 4	10310
5	3	Jenny	Ulica Ruža 1	51000
6	4	Bolero	Ulica kneza Branimira 19	10000
7	5	Kreacija	Južna ulica 55	51000
8	6	As	Dubečka ulica 21	10000

Slika 15 Prikaz naziva krojačnice čija je šifra 32 nakon ažuriranja

U primjeru je korištena DML naredba UPDATE za tablicu KROJACNICA, odnosno za krojačnicu čija je šifra 32, ažurira se naziv adrese i ukoliko je ažuriranje uspješno ispiše se pripadajuća poruka: *Adresa je uspješno ažurirana za krojačnicu čija je šifra: 32*

Za atribut korišten je %NOTFOUND koji vraća TRUE kada naredba UPDATE neće ažurirati stupac *adresa.* Ukoliko se unese kriva šifra krojačnice ispiše se pripadajuća poruka: *Ne postoji krojačnica pod šifrom 77. Promijeni unos!!*

7.1.2 Eksplicitni kursor

Eksplicitni kursor je kursor kojeg program deklarira putem PL/SQL koda za izvršenje bilo kojeg SELECT upita koji vraća više od jednog retka. Deklarira se u deklarativnom dijelu PL/SQL bloka. Za rad s eksplicitnim kursorom koriste se naredbe OPEN, FETCH i CLOSE. Rad s eksplicitnim kursorom uključuje sljedeće korake:

- deklariranje (eng. *declaring*) kursora za inicijalizaciju memorije
- otvaranje (eng. *opening*) kursora za dodjelu memorije
- dohvaćanje (eng. *fetching*) kursora za dohvaćanje podataka
- zatvaranje (eng. *closing*) kursora za oslobađanje alocirane memorije

Deklariranje kursora

Eksplicitni kursor mora biti deklariran u deklarativnom dijelu PL/SQL bloka ili paketa prije nego što se može koristiti:

CURSOR *ime_kursora* IS SELECT *upit*;

ime_kursora može biti bilo koji valjani identifikator, dok *upit* može biti bilo koja SELECT naredba.

Otvaranje kursora

Prije samog korištenja kursora potrebno ga je otvoriti. Otvaranjem kursora, kursoru se dodjeljuje memorija za dohvaćanje redaka koju vraća SQL naredba. Za otvaranje kursora koristi se sintaksa:

OPEN *ime_kursora*;

Naredba OPEN izvršava naredbu koja je pridružena kursoru, identificira skup rezultata (eng. *Result set*) te pozicionira kursor ispred prvog retka. Kursor u isto vrijeme može imati samo jedan aktivan zapis.

Dohvaćanje kursora

Nakon otvaranja eksplicitnog kursora, naredba FETCH dohvaća trenutni redak skupa rezultata, pohranjuje vrijednost retka tog stupca u varijablu ili zapis te pomiče kursor na sljedeći redak. Da bi se dohvatili svi redci u skupu rezultata, potrebno je dohvatiti sve do posljednjeg retka. Sintaksa za dohvaćanje kursora je:

FETCH *ime_kursora* INTO *ime_varijable;*

ime_varijable može biti jedna ili više varijabli razdvojenih zarezom koje odgovaraju broju i tipu podatka stupaca uključenih u skup rezultata. Atributi %ROWTYPE i %TYPE korisni su za deklariranje varijabli i zapisa u FETCH naredbama. Naredba FETCH obično se koristi unutar LOOP naredbe koja se prekida kada nema redaka za dohvaćanje. Za provjeru da li je dohvaćanje kursora bilo uspješno ili ne koriste se atributi %FOUND i %NOTFOUND.

Zatvaranje kursora

Kada se obradi i posljednji redak, naredbom CLOSE zatvara se kursor kako bi se oslobodila alocirana memorija. Nakon završetka prethodnog koraka, ukoliko neki kursor nije zatvoren Oracle to radi automatski ali taj način zatvaranje se ne preporučuje [29] [30] [31]. Sintaksa za zatvaranje kursora je:

CLOSE *ime_kursora;*

Primjer koda za model krojačnice:

```
DECLARE
        CURSOR cursor_ime_krojaca IS
                SELECT ime_krojaca,
                        prezime_krojaca,
                        naziv_krojacnice,
                        adresa
                FROM   krojac a,
                        krojacnica b
                WHERE  a.sifra_krojacnice = b.sifra_krojacnice;
        var_krojac_zapis cursor_ime_krojaca%ROWTYPE;
BEGIN
        OPEN cursor_ime_krojaca;
        LOOP
                FETCH cursor_ime_krojaca INTO var_krojac_zapis;
                EXIT WHEN cursor_ime_krojaca%NOTFOUND;
```

```
                DBMS_OUTPUT.PUT_LINE('Ime krojača:  '
                    || var_krojac_zapis.ime_krojaca
                    || ' '
                    || var_krojac_zapis.prezime_krojaca
                    || '   **   Naziv krojacnice: '
                    || var_krojac_zapis.naziv_krojacnice
                    || '   **   Adresa krojacnice: '
                    || var_krojac_zapis.adresa);
        END LOOP;
        CLOSE cursor_ime_krojaca;
END;
 /
```

Rezultat:

*Ime krojača: Mirjana Tarnik   **   Naziv krojacnice: Botun   **   Adresa krojacnice: Vukovarska 8*
*Ime krojača: Marko Glog   **   Naziv krojacnice: Igla   **   Adresa krojacnice: Savska 14*
*Ime krojača: Barbara Lovrić   **   Naziv krojacnice: Botun   **   Adresa krojacnice: Vukovarska 8*
*Ime krojača: Melita Ožetski   **   Naziv krojacnice: Stil   **   Adresa krojacnice: Dolanec 37*

PL/SQL procedure successfully completed.

U primjeru su dohvaćeni ime i prezime krojača te naziv i adresa krojačnice u kojoj rade. Na početku definirao se je naziv kursora. Pomoću naredbe SELECT dohvaćene su željene vrijednosti iz obiju tablica. Nakon toga potrebno je otvoriti kursor te ulaskom u petlju i pomoću naredbe FETCH dohvaćaju se svi redci u skupu rezultata. Na samom kraju potrebno je zatvoriti kursor naredbom CLOSE.

7.1.3 Kursori s FOR petljom

Kursori se obično koriste s petljama. Kursor s FOR petljom je modifikacija obične FOR petlje. To je petlja koja je namijenjena kursoru i koja automatski provjerava broj redaka i izlazi se iz petlje kada se svi podaci koji su pohranjeni unutar petlji počinju ponavljati. Proces otvaranja, dohvaćanja i zatvaranja radi se implicitno. Kursor s FOR petljom automatski radi sljedeće:

- Indeks petlje implicitno je deklariran kao varijabla zapisa %ROWTYPE

- Otvara kursor
- Dohvaća zapis iz kursora za svaku iteraciju
- Zatvara kursor nakon obrade svih zapisa
- Kursor se može zatvoriti korištenjem naredbi EXIT i GOTO

Sintaksa kursora s FOR petljom je:

```
FOR ime_varijable IN ime_kursora
LOOP
        izvršne_naredbe;
END LOOP;
```

Primjer koda za model krojačnice:

```
DECLARE
   CURSOR cur_klijent_popravci IS
    SELECT ime_klijenta,
                prezime_klijenta,
                broj_popravaka,
                cijena_popravaka
    FROM   klijent k
                JOIN popravlja p
                      ON p.sifra_klijenta = k.sifra_klijenta
    WHERE  ime_klijenta = 'Katarina'
                AND broj_popravaka > 2;
BEGIN
   FOR cru_klijent_zapis IN cur_klijent_popravci LOOP
     DBMS_OUTPUT.PUT_LINE('Ime klijenta: '
                    ||cru_klijent_zapis.ime_klijenta
                    ||' '
                    ||cru_klijent_zapis.prezime_klijenta
                    ||'   **   Broj popravaka: '
                    ||cru_klijent_zapis.broj_popravaka
                    ||'   **   Cijena popravaka: '
                    ||cru_klijent_zapis.cijena_popravaka
                    ||' kuna ');
   END LOOP;
END;
/
```

Rezultat:

*Ime klijenta: Katarina Meštrović ** Broj popravaka: 4 ** Cijena popravaka: 180 kuna*

PL/SQL procedure successfully completed.

7.1.4 Kursori s parametrima

Parametri se mogu proslijediti unutar kursora i koristiti u naredbi. Definiran je samo tip podataka parametra. Opcionalno parametru se može zadati i DEFAULT vrijednost u slučaju kada niti jedna vrijednost nije proslijeđena kursoru. Sintaksa za kursor s parametrima je:

CURSOR *ime_kursora* (*ime_varijable tip_podatka*) IS SELECT *naredba*;

U slučaju više parametara, parametri i odgovarajući parametri odvajaju se zarezom [32]. Bitno je još naglasiti da se kursori mogu ugnijezditi unutar SQL naredbe. Ugniježđeni kursori omogućavaju iteraciju nad podacima u različitim fazama. Ugniježđeni kursori ne mogu se koristiti kod implicitnih kursora.

Primjer koda za model krojačnice:

```
DECLARE
   CURSOR cur_krojacnice_mjesto (postanski_broj NUMBER) IS
      SELECT k.naziv_krojacnice, m.naziv_mjesta
      FROM krojacnica k
      JOIN mjesto m on m.postanski_broj = k.postanski_broj
      WHERE m.postanski_broj = k.postanski_broj;
BEGIN
   FOR cru_detalji IN cur_krojacnice_mjesto(1) LOOP
      DBMS_OUTPUT.PUT_LINE('Naziv krojacnice: '
                                    || cru_detalji.naziv_krojacnice
                                    ||'   **   '
                                    || 'Naziv mjesta: '
                                    || cru_detalji.naziv_mjesta);
   END LOOP;
END;
/
```

Rezultat:

*Naziv krojacnice: Igla ** Naziv mjesta: Zagreb*
*Naziv krojacnice: Botun ** Naziv mjesta: Rijeka*
*Naziv krojacnice: Stil ** Naziv mjesta: Ivanić Grad*

PL/SQL procedure successfully completed.

Deklarira se kursor koji prihvaća parametar *postanski_broj*. Kursor vraća popis mjesta u kojima se nalaze krojačnice.

Primjer koda za ugniježđeni kursor za model krojačnice:

```
DECLARE
    krojacnica_kljuc krojacnica.postanski_broj%TYPE;
 CURSOR cursor_mjesto IS
    SELECT *
    FROM mjesto
    ORDER BY naziv_mjesta;
 CURSOR cursor_krojac IS
    SELECT *
    FROM krojacnica
    WHERE postanski_broj = krojacnica_kljuc;
BEGIN
        FOR cursor_mje IN cursor_mjesto
        LOOP
            krojacnica_kljuc := cursor_mje.postanski_broj;
            DBMS_OUTPUT.PUT_LINE('----------------------------------');
            DBMS_OUTPUT.PUT_LINE('Naziv mjesta : '||cursor_mje.naziv_mjesta);
            DBMS_OUTPUT.PUT_LINE('************************');
                FOR cursor_krojacnica IN cursor_krojac
                LOOP
                        DBMS_OUTPUT.PUT_LINE('Naziv krojačnice:
'||cursor_krojacnica.naziv_krojacnice);
        END LOOP;
   END LOOP;
END;
 /
```

Rezultat:

```
---------------------------------
Naziv mjesta : Ivanić Grad
************************
```

Naziv krojačnice: Švelje
Naziv krojačnice: Stil

Naziv mjesta : Rijeka
****
Naziv krojačnice: Botun

Naziv mjesta : Zagreb
****
Naziv krojačnice: Igla

PL/SQL procedure successfully completed.

Primjer prikazuje ugniježđeni kursor kod kojega kursor *cursor_mjesto* sprema sve nazive mjesta za koja se dohvaćaju redci koji sadrže nazive krojačnica koje se nalaze u određenom mjestu na temelju poštanskog broja.

7.2 Kompozitni tipovi podataka

Kompozitni tipovi podataka (eng. *Composite Data Types*) razlikuju se od skalarnih po tome što pohranjuju vrijednosti unutarnjih komponenti. Kao parametre, u potprograme mogu se proslijediti kompozitne varijable[13], a unutarnjim komponentama može se pristupiti pojedinačno. Unutarnje komponente mogu biti skalarne ili kompozitne. Kompozitni tipovi podataka uključuju zapise (eng. *Records*), ugniježđene tablice (eng. *Nested Tables*), asocijativni niz (eng. *Associative Array*/*Index-By Tables*) i polja varijabilne duljine (eng. *Variable-Size Arrays*/*Varrays*).

7.2.1 Zapis

Zapis (eng. *Record*) sastavljen je od jedne ili više komponenti ili elemenata koji su pohranjeni u polje od kojih svako polje ima svoju vrijednost odnosno ime i tip podatka. Prema zadanim postavkama, početna vrijednost polja je NULL. Isto tako za polje se može postaviti

[13] Varijable koje spremaju vrijednost unutarnjih komponenti.

NOT NULL ograničenje, ali prvo se mora navesti početna vrijednost koja nije NULL (u suprotnom bez NOT NULL ograničenja, početna vrijednost koja nije NULL nije obavezna).

PL/SQL ima tri tipa zapisa:

- Zapis temeljen na tablici (eng. *Table-based record*)
- Zapis temeljen na kursoru (eng. *Cursor-based record*)
- Programerski definiran zapis (eng. *Programmer-defined record*)

Prije korištenja zapisa potrebno ga je deklarirati. Zapis se definira i deklarira u deklarativnom dijelu bloka ili u specifikaciji paketa. Tip zapisa može se definirati implicitno ili eksplicitno. Kod implicitne deklaracije koristi se postojeća tablica, pogled ili kursor kao referenca te atribut %ROWTYPE. Ova metoda osigurava sinkronizaciju s bazom podataka. Nedostatak je što se mora unijeti cijeli zapis iako je potrebno samo nekoliko stupaca. Kod eksplicitne deklaracije prvotno se definira tip podataka i zatim se kreira varijabla tog tipa.
Atribut %TYPE koristi se kod eksplicitno definiranih zapisa.

7.2.1.1 Zapis temeljen na tablici

Zapis temeljen na tablici koristi %ROWTYPE atribut s imenom tablice prilikom deklaracije u kojem svako polje odgovara stupcu u tablici. Sintaksa za zapis temeljen na tablici je:

```
DECLARE
    naziv_zapisa ime_tablice%ROWTYPE;
```

Primjer koda za model krojačnice:

```
DECLARE
    v_tkanina tkanina%ROWTYPE;
BEGIN
    SELECT *
    INTO   v_tkanina
    FROM   tkanina
    WHERE  sifra_tkanine = 7;
    DBMS_OUTPUT.PUT_LINE('Podaci o tkanini :   Šifra tkanine: '
        ||v_tkanina.sifra_tkanine
        || '  **  '
```

```
                    ||'  Naziv tkanine: '
                    ||v_tkanina.naziv_tkanine
                    || '  **  '
                    ||'   Boja tkanine: '
                    ||v_tkanina.boja_tkanine );
END;
/
```

Rezultat:

*Podaci o tkanini : Šifra tkanine: 7 ** Naziv tkanine: Svila ** Boja tkanine: Crna*

PL/SQL procedure successfully completed.

7.2.1.2 Zapis temeljen na kursoru

Zapis temeljen na kursoru isto tako koristi atribut %ROWTYPE s eksplicitnim kursorom ili s varijablom kursora u kojoj svako polje odgovara stupcu ili aliasu u SELECT izrazu kursora. Sintaksa glasi:

```
DECLARE
        naziv_zapisa ime_kursora%ROWTYPE;
```

Primjer koda za model krojačnice:

```
DECLARE
   CURSOR cursor_cjenikUsluga IS
      SELECT *
      FROM cjenik_usluga WHERE cijena_usluge BETWEEN 45 AND 100
      ORDER BY cijena_usluge, naziv_usluge;
   cursor_zapis cursor_cjenikUsluga%ROWTYPE;
BEGIN
    OPEN cursor_cjenikUsluga;
     LOOP
     FETCH cursor_cjenikUsluga into cursor_zapis;
      EXIT WHEN cursor_cjenikUsluga%NOTFOUND;
          DBMS_OUTPUT.PUT_LINE('Šifra usluge: '
                    ||cursor_zapis.sifra_usluge
                    || '  **   '
                    ||' Naziv usluge: '
                    ||cursor_zapis.naziv_usluge
                    || '  **   '
                    ||' Cijena usluge: '
```

```
                ||cursor_zapis.cijena_usluge
                || ' kuna');
    END LOOP;
CLOSE cursor_cjenikUsluga;
END;
/
```

Rezultat:

*Šifra usluge: 3 ** Naziv usluge: Strojno skraćivanje hlača ** Cijena usluge: 45 kuna*
*Šifra usluge: 1 ** Naziv usluge: Šivanje ** Cijena usluge: 45 kuna*
*Šifra usluge: 8 ** Naziv usluge: Sužavanje suknji ** Cijena usluge: 70 kuna*
*Šifra usluge: 7 ** Naziv usluge: Produljivanje rukava ** Cijena usluge: 80 kuna*
*Šifra usluge: 5 ** Naziv usluge: Sužavanje haljine ** Cijena usluge: 90 kuna*

PL/SQL procedure successfully completed.

7.2.1.3 Programerski definiran zapis

Za definiranje zapisa koji se ne temelji na tablici ili na kursoru PL/SQL pruža mogućnost korištenja programerskog definiranog zapisa. Koriste se TYPE…RECORD izrazi gdje je svako polje eksplicitno definirano sa svojim imenom i tipom podatka u TYPE izrazu za taj zapis. Sintaksa glasi:

```
TYPE ime_tipa IS RECORD (
        naziv_polja1 tip_podataka1 [[NOT NULL] :=| DEFAULT zadana_vrijednost],
        naziv_polja2 tip_podataka2 [[NOT NULL] :=| DEFAULT zadana_vrijednost],
        …
        naziv_poljaN tip_podatakaN [[NOT NULL] :=| DEFAULT zadana_vrijednost]
        );
```

gdje se korištenjem TYPE izraza definira struktura zapisa, odnosno *ime_tipa* predstavlja ime tipa zapisa koje će se koristiti prilikom deklaracije stvarnog zapisa u drugom koraku. *naziv_poljaN* označava naziv N-tog polja u zapisu, a *tip_podatakaN* je tip podataka tog N-tog polja. Opcionalno može se dodati NOT NULL ograničenje ili se dodijeli zadana vrijednost. Za tipove podataka polja zapisa mogu se koristiti skalarni tipovi podataka (BINARY_FLOAT, CHAR, DATE i tako dalje), podtipovi, kursori, kolekcije, usidreni tipovi podataka (%TYPE i %ROWTYPE atributi) [7] [33] [10].

Primjer koda za model krojačnice:

```
DECLARE
        TYPE programerski_tip IS RECORD(
                ime_klijenta klijent.ime_klijenta%TYPE,
                prezime_klijenta klijent.prezime_klijenta%TYPE,
                spol_klijenta  klijent.spol_klijenta%TYPE);
        polje programerski_tip;
         var_sifra_klijenta klijent.sifra_klijenta%TYPE := 357;
BEGIN
        SELECT ime_klijenta,
        prezime_klijenta,
        spol_klijenta
        INTO polje
        FROM klijent
        WHERE sifra_klijenta = var_sifra_klijenta;
        DBMS_OUTPUT.PUT_LINE('Ime klijenta: '
                                        || polje.ime_klijenta
                                        || ' '
                                        || polje.prezime_klijenta
                                        || '  **  '
                                        || ' Spol: '
                                        || polje.spol_klijenta);
END;
/
```

Rezultat:

*Ime klijenta: Marijan Horvat ** Spol: Muško*

PL/SQL procedure successfully completed.

Na početku definiralo se *ime_tipa* koje sadrži tri polja. Unutar SELECT klauzule sprema se ime, prezime i spol klijenta pod šifrom 357.

Operacije na razini zapisa koje podržava PL/SQL su:

- NULL vrijednost može se dodijeliti zapisu koji ima jednostavan zadatak
- Sadržaj jednog zapisa može se kopirati u drugi zapis, sve dok imaju isti broj polja i tip podataka
- Zapis se može vratiti nazad kroz sučelje funkcije

- Zapis se može definirati i proslijediti kao argument u popisu parametara

7.2.1.4 Ugniježđeni zapisi

PL/SQL omogućuje definiranje ugniježđenih zapisa (eng. *Nested Records*). Ugniježđeni zapis uključuje zapis kao polje unutar drugog zapisa. Zapis koji sadrži ugniježđeni zapis kao polje naziva se *enclosing record.* Ugniježđeni zapis moćan je alat za strukturiranje podataka programa i skrivanje složenosti koda [1].

Primjer koda za model krojačnice:

```
DECLARE
        TYPE mjesto_zapis IS RECORD
                (postanski_broj NUMBER(5)
                ,naziv_mjesta VARCHAR2(30));

        TYPE krojacnica_zapis IS RECORD
                (sifra_krojacnice NUMBER(4)
                ,naziv_krojacnice VARCHAR2(30 CHAR)
                ,adresa  VARCHAR2(30)
                ,postanski_broj NUMBER(5));

        TYPE mjesto_krojacnica_zapis IS RECORD
                (krojacnica KROJACNICA_ZAPIS
                ,mjesto MJESTO_ZAPIS);

        mjesto_krojacnica Mjesto_Krojacnica_Zapis;

BEGIN
                mjesto_krojacnica.mjesto.postanski_broj := 21000 ;
                mjesto_krojacnica.mjesto.naziv_mjesta := 'Split';

                mjesto_krojacnica.krojacnica.sifra_krojacnice := 589;
                mjesto_krojacnica.krojacnica.naziv_krojacnice := 'Brzi popravci';
                mjesto_krojacnica.krojacnica.adresa := 'Bajamontijeva ulica 5';
                mjesto_krojacnica.krojacnica.postanski_broj := 21000 ;

                INSERT
                INTO mjesto
                VALUES
                        (mjesto_krojacnica.mjesto.postanski_broj
                        ,mjesto_krojacnica.mjesto.naziv_mjesta);
```

```
        INSERT
        INTO krojacnica
        VALUES
            (mjesto_krojacnica.krojacnica.sifra_krojacnice
            ,mjesto_krojacnica.krojacnica.naziv_krojacnice
            ,mjesto_krojacnica.krojacnica.adresa
            ,mjesto_krojacnica.krojacnica.postanski_broj);
        COMMIT;
    END;
    /
```

Rezultat:

	SIFRA_KROJACNICE	NAZIV_KROJACNICE	ADRESA	POSTANSKI_BROJ
1	32	Igla	Savska 5	10000
2	214	Botun	Vukovarska 8	51000
3	7	Stil	Dolanec 37	10310
4	1	Šivaona	Opatinec 4	10310
5	3	Jenny	Ulica Ruža 1	51000
6	4	Bolero	Ulica kneza Branimira 19	10000
7	5	Kreacija	Južna ulica 55	51000
8	6	As	Dubečka ulica 21	10000

Slika 16 Prikaz tablice KROJACNICA prije naredbe INSERT

	POSTANSKI_BROJ	NAZIV_MJESTA
1	10000	Zagreb
2	51000	Rijeka
3	10310	Ivanić Grad

Slika 17 Prikaz tablice MJESTO prije naredbe INSERT

	SIFRA_KROJACNICE	NAZIV_KROJACNICE	ADRESA	POSTANSKI_BROJ
1	32	Igla	Savska 5	10000
2	214	Botun	Vukovarska 8	51000
3	7	Stil	Dolanec 37	10310
4	1	Šivaona	Opatinec 4	10310
5	3	Jenny	Ulica Ruža 1	51000
6	4	Bolero	Ulica kneza Branimira 19	10000
7	5	Kreacija	Južna ulica 55	51000
8	6	As	Dubečka ulica 21	10000
9	589	Brzi popravci	Bajamontijeva ulica 5	21000

Slika 18 Prikaz tablice KROJACNICA nakon naredbe INSERT

	POSTANSKI_BROJ	NAZIV_MJESTA
1	10000	Zagreb
2	51000	Rijeka
3	10310	Ivanić Grad
4	21000	Split

Slika 19 Prikaz tablice MJESTO nakon naredbe INSERT

Za primjer ugniježđenog zapisa koristile su se tablice čija su polja eksplicitno definirana unutar PL/SQL programa. Na početku definirala su se dva tipa zapisa *mjesto_zapis* i *krojacnica_zapis.* Nakon toga tip zapisa treba biti definirana od dva tipa zapisa koje je odredio programer. Za dvije odabrane tablice potrebno je inicijalizirati nove vrijednosti za zapis. Nakon toga naredbom INSERT unesu se nove vrijednosti.

7.2.2 Kolekcije

PL/SQL kolekcije su strukture podataka koje funkcioniraju poput listi[14] ili jednodimenzionalnog niza, odnosno kolekcije su strukture za upravljanje više redova podataka. Elementi koji se nalaze unutar liste istog su tipa podataka. Svakom elementu uređene liste pristupa se pomoću jedinstvenog indeksa[15], a kod neuređene liste svakom elementu pristupa se pomoću jedinstvenog identifikatora koji može biti broj, *hash* vrijednost ili ime niza. Kolekcije su iznimno korisne kada se velike količine podataka istog tipa moraju obraditi ili kada se njima mora manipulirati. Oracle podržava tri različita tipa kolekcija [1] [9] [10] [34]:

- Asocijativni niz (eng. *Associative arrays* također poznat kao *Index-by tables*)
- Ugniježđene tablice (eng. *Nested tables*)
- Polja varijabilne duljine (eng. *Variable-size array* također poznata kao *Varrays*)

[14] Lista može biti uređena ili neuređena.
[15] Kod PL/SQL-a indeksi kreću od broja 1.

7.2.2.1 Asocijativni niz

Asocijativni niz je tip kolekcije koji povezuje jedinstveni ključ (svaki ključ je jedinstveni indeks) s vrijednošću pomoću sintakse *ime_varijable(indeks).* Indeks može koristit dva tipa podataka, a to su znakovni tipovi podataka (VARCHAR2, VARCHAR, STRING ili LONG) ili brojevi (PLS_INTEGER). Potrebno je naglasiti da se indeksi pohranjuju redoslijedom sortiranja, a ne redoslijedom kreiranja. Asocijativni niz sadrži elemente[16] s istim tipom podataka koje nazivamo **homogenim elementima** te broj elemenata niza nema fiksno ograničenje nego broj elemenata dinamički raste kako se dodavaju novi elementi. Ne može se koristiti u tablicama, nego samo kao programska struktura unutar PL/SQL-a. Deklaracija se odvija u dva koraka. U prvom koraku deklarira s tip asocijativnog niza, a u drugom koraku deklarira se varijabla asocijativnog niza tog tipa. Sintaksa glasi [35]:

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

gdje *ime_tipa* predstavlja ime tipa asocijativnog niza koje se koristi u drugom koraku kako bi se deklarirala tablica. *tip_elemenata* predstavlja tip podataka elemenata niza koji može biti NUMBER, VARCHAR2 ili DATE te prema kojemu će se vršiti pretraživanje kolekcije i dohvaćanje vrijednosti. *tip_indeksa* predstavlja tip podataka indeksa koji se koristi za organiziranje elemenata u nizu.

Primjer koda za krojačnicu:

```
DECLARE
        CURSOR cursor_tkanine IS
                SELECT
                naziv_tkanine
                FROM tkanina
                WHERE rownum<10;
        TYPE tkanine IS TABLE OF tkanina.naziv_tkanine%TYPE
                INDEX BY BINARY_INTEGER;
        tablica_tkanine tkanine;
        brojac INTEGER :=0;
BEGIN
        FOR tkanine_zapis IN cursor_tkanine LOOP
                brojac := brojac + 1;
```

[16] Asocijativan niz je prazan (ali nije NULL) sve dok se ne popuni.

```
            tablica_tkanine(brojac) := tkanine_zapis.naziv_tkanine;
            DBMS_OUTPUT.PUT_LINE('Naziv tkanine('
                                    ||brojac
                                    ||'): '
                                    ||tablica_tkanine(brojac));
        END LOOP;
END;
/
```

Rezultat:

Naziv tkanine(1): Svila
Naziv tkanine(2): Pamuk
Naziv tkanine(3): Vuna
Naziv tkanine(4): Batist
Naziv tkanine(5): Čipka
Naziv tkanine(6): Krzno
Naziv tkanine(7): Pliš
Naziv tkanine(8): Platno
Naziv tkanine(9): Saten

PL/SQL procedure successfully completed.

U primjeru asocijativni tip *tablica_tkanine* popunjen je nazivima tkanina iz tablice TKANINE koji su dohvaćeni kursorom. Varijabla *brojac* koristila se za kao indeks za referenciranje pojedinačnih elemenata tablice.

7.2.2.2 Ugniježđene tablice

Ugniježđene tablice su tip kolekcija koje su jednodimenzionalno polje[17] s proizvoljnim brojem elemenata te imaju elemente istog tipa podataka poput asocijativnog niza. To ih razlikuje od nizova. Mogu se koristiti za definiranje tablica, zapisa i objekata te unutar SQL-a i PL/SQL-a. Ugniježđene tablice su multiskupovi što znači da elementi unutar tablice nemaju svojstveni redoslijed. Kada se dohvate vrijednosti ugniježđene tablice iz baze podataka i kada se pohrane u varijablu ugniježđene tablice, PL/SQL redcima dodjeljuje uzastopne indekse koji počinju od broja 1, te se pomoću njih može pristupiti pojedinačnim redcima varijable

[17] Jednodimenzionalno polje znači da svaki redak ima jedan stupac podataka poput jednodimenzionalnog niza.

ugniježđene tablice. Sintaksa glasi *ime_varijable(indeks)*. Potrebno je naglasiti da indeksi i redoslijed redaka ugniježđene tablice možda neće ostat nepromijenjeni zato što se ugniježđena tablica pohranjuje i dohvaća iz baze podataka. Deklaracije se odvija u dva koraka. U prvom koraku deklarira se tip ugniježđene tablice, a u drugom koraku deklarira se varijabla ugniježđene tablice na temelju tipa ugniježđene tablice. Sintaksa glasi:

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

Deklariranje je slično kao i kod asocijativnog niza, s razlikom što se kod ugniježđenih tablica ne koristi klauzula INDEX BY. Potrebno je napomenuti da ako se ugniježđena tablica ne inicijalizira, tada vrijednost varijable ugniježđene tablice sadrži NULL vrijednost, a da bi se to izbjeglo potrebno ju je inicijalizirati na način da se učini praznom ili da joj se dodijeli vrijednost koja nije NULL, odnosno koristi se konstruktor. Za stvaranje konstruktora koristi se sintaksa:

ime_varijable := *tip_podataka();*

Ugniježđena tablica može se deklarirati i inicijalizirati u jednom koraku koristeći sintaksu:

ime_varijable tip_podataka := *tip podataka();*

Za povećanje prostora za jedan ili više elemenata u ugniježđenoj tablici koristi se naredba EXTEND:

ime_varijable.EXTEND;

Nakon toga pomoću operatora dodjele, ugniježđenoj tablici dodaju se element koristeći sintaksu:

ime_varijable := *element;*

Metoda EXTEND ne može se koristiti s asocijativnim nizovima.

Primjer koda za model krojačnice:

```
DECLARE
    CURSOR cursor_ime_klijenta IS
        SELECT ime_klijenta
        FROM klijent
        FETCH FIRST 10 ROWS ONLY;

    TYPE klijent_ime_klijenta_tip IS TABLE OF klijent.ime_klijenta%TYPE;
    tip_ime_klijenta klijent_ime_klijenta_tip := klijent_ime_klijenta_tip();

BEGIN
```

```
        FOR ime_klijenta_kursor IN cursor_ime_klijenta
        LOOP
                tip_ime_klijenta.EXTEND;
                tip_ime_klijenta(tip_ime_klijenta.LAST) :=
ime_klijenta_kursor.ime_klijenta;
        END LOOP;

        FOR indeks_klijent IN tip_ime_klijenta.FIRST..tip_ime_klijenta.LAST
        LOOP
                DBMS_OUTPUT.PUT_LINE('Ime klijenta: ' ||
tip_ime_klijenta(indeks_klijent));
        END LOOP;
END;
/
```

Rezultat:

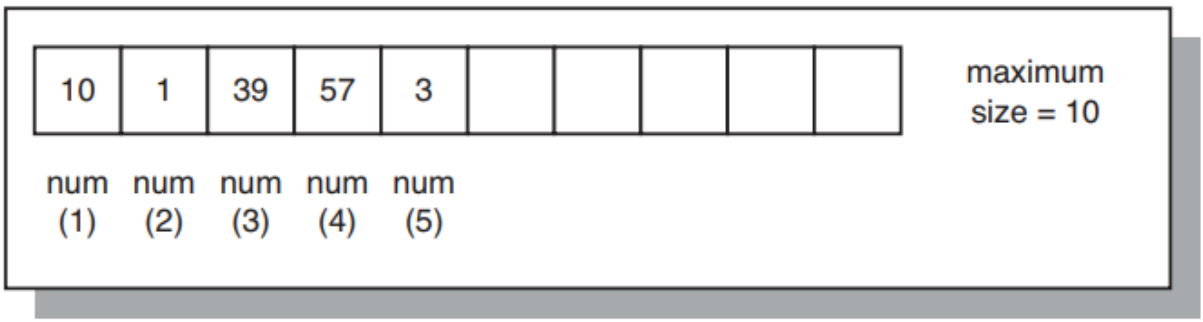
Ime klijenta: Katarina
Ime klijenta: Marijan
Ime klijenta: Vlado
Ime klijenta: Danijela
Ime klijenta: Danijel
Ime klijenta: Drinka
Ime klijenta: Blaž
Ime klijenta: Stanija
Ime klijenta: Nadia
Ime klijenta: Bernada

PL/SQL procedure successfully completed.

U primjeru se pomoću kursora dohvatilo prvih deset imena klijenata i ta se imena dodaju ugniježđenoj tablici. Definira se tip ugniježđene tablice *klijent_ime_klijenta_tip*, deklarira se varijabla ugniježđene tablice *tip_ime_klijenta klijent_ime_klijenta_tip* i konstruktor *klijent_ime_klijenta_tip()*. Nakon toga dohvaća se ime klijenata iz kursora koje se doda ugniježđenoj tablici. Pomoću LOOP petlje izvodi se iteracija elemenata ugniježđene tablice.

7.2.2.3 Polja varijabilne duljine

Tip kolekcija koje su jednodimenzionalno polje koje ima fiksni maksimalni broj elemenata[18] s istim tipom podataka naziva se polje varijabilne duljine. Mogu se koristit za definiranje tablica, zapisa, objekata i može im se pristupiti unutar SQL-a i PL/SQL-a. Za pristupanje elementima varijable koristi se sintaksa *ime_varijable(indeks)*. Vrijednost indeksa počinje od broja 1 koji označava donju granicu, te se povećava za jedan sve dok ne dosegne gornju granicu koja predstavlja trenutni broj elemenata. Gornja granica mijenja se s obzirom da li se elementi dodaju ili brišu i maksimalan broj elemenata ne može se premašiti.



Slika 20 Prikaz polja varijabilne duljine koje se sastoji od pet cijelih brojeva [1]

Za razliku od ugniježđenih tablica, redoslijed elemenata i indeksa ostaje sačuvan prilikom pohranjivanja i dohvaćanja polja varijabilne duljine iz baze podataka. Kako se taj proces odvija istovremeno, korištenje polja varijabilne duljine moglo bi biti nepraktično kod velikog broja elemenata. Za deklariranje koristi se sintaksa:

TYPE *ime_tipa* IS VARRAY{VARRAY | VARYING ARRAY} (*max_br_elemenata*)
OF *tip_elemenata* [NOT NULL];
ime_varijable IME_TIPA;

gdje je struktura polja varijabilne duljine definirana TYPE izrazom, *ime_tipa* označava tip polja varijabilne duljine. Vrijedi napomenuti da postoje dvije varijacije tipa VARRAY i VARYING ARRAY. *max_br_elemenata* označava maksimalan broj elemenata (gornju granicu) koji je dopušten. *tip_elemenata* označava tip elemenata varijable tipa polje varijabilne duljine. Slično kao i kod ugniježđenih tablica, ako se polje varijabilne duljine ne inicijalizira prilikom deklaracije tada sadrži NULL vrijednosti. Kako bi se to izbjeglo

[18] Za razliku od asocijativnog niza i ugniježđenih tablica.

potrebno ga je inicijalizirati na način da se učini praznim ili da mu se dodijeli vrijednost koja nije NULL, odnosno da se koristi konstruktor. Sintaksa za deklaraciju je:

ime_varijable tip_podataka [:= *tip_podataka(...)*];

gdje *ime_varijable* označava ime varijable polja varijabilne duljine. *tip_podataka* označava tip polja varijabilne duljine, a *tip_podataka(...)* je konstruktor tipa polja varijabilne duljine koji prihvaća elemente koji su odvojeni zarezom kao argument. Ima isto ime kao tip polja varijabilne duljine.

Primjer koda za model krojačnice:

```
DECLARE
        TYPE zapis_krojac_tip IS RECORD(
                ime_krojaca krojac.ime_krojaca%TYPE,
                prezime_krojaca krojac.prezime_krojaca%TYPE,
                 iznos_place krojac.iznos_place%TYPE);

         TYPE krojac_tip IS VARRAY(5)  OF zapis_krojac_tip;

        tip_krojac krojac_tip := krojac_tip();

        CURSOR cursor_krojac IS
        SELECT ime_krojaca, iznos_place, prezime_krojaca
        FROM krojac
        ORDER BY iznos_place DESC
        FETCH FIRST 5 ROWS ONLY;
BEGIN
   FOR zapis_krojac IN cursor_krojac LOOP
        tip_krojac.EXTEND;
        tip_krojac(tip_krojac.LAST).ime_krojaca := zapis_krojac.ime_krojaca;
        tip_krojac(tip_krojac.LAST).prezime_krojaca :=
zapis_krojac.prezime_krojaca;
        tip_krojac(tip_krojac.LAST).iznos_place  := zapis_krojac.iznos_place;
   END LOOP;

   FOR indeks_krojac IN tip_krojac .FIRST..tip_krojac.LAST
   LOOP
      DBMS_OUTPUT.PUT_LINE('Krojač '
                ||tip_krojac(indeks_krojac).ime_krojaca
                || ' '
                || tip_krojac(indeks_krojac).prezime_krojaca
                ||  ' ima plaću u iznosu od '
```

```
                    ||tip_krojac(indeks_krojac).iznos_place || ' kune');
        DBMS_OUTPUT.PUT_LINE('-------------------------------------------------------');
    END LOOP;
END;
/
```

Rezultat:

Krojač Mislav Danolić ima plaću u iznosu od 7800 kune

Krojač Božidar Tomiek ima plaću u iznosu od 6820 kune

Krojač Nives Šalek ima plaću u iznosu od 6000 kune

Krojač Barbara Lovrić ima plaću u iznosu od 5500 kune

Krojač Stankica Šrenk ima plaću u iznosu od 5100 kune

PL/SQL procedure successfully completed.

U primjeru ponovno se koristio kursor kao i kod ugniježđenih tablica ali ovdje se koristi da se dohvate imena pet krojača s plaćom koja se kreće od najveće do najmanje plaće iz tablice KROJAC, te se dohvaćene vrijednosti prosljeđuju polju varijabilne duljine. Na početku potrebno je deklarirati tip zapisa, tipa varijabilnog polja koji sadrži zapise pet elemenata (imena krojača). Deklarira se kursor koji vraća pet imena pet krojača s najvećom plaćom. Pomoću LOOP petlje izvodi se iteracija elemenata.

7.2.3 Metode kolekcija

Metode kolekcija ugrađene su funkcije i procedure unutar PL/SQL-a koje omogućuju prikupljanje informacija o nečemu i izmjenu sadržaja kolekcija. Čine kolekcije lakšima za korištenje i olakšavaju održavanje aplikacija. Za pozivanje pojedinih metoda koristi se točka:

ime_kolekcija.ime_metode

Tablica 6 prikazuje metode kolekcija.

Metoda	Tip	Opis
EXISTS	Funkcija	Vraća TRUE ako i samo ako navedeni element postoji unutar ugniježđene tablice ili

		polja varijabilne duljine. Inače vraća FALSE
COUNT	Funkcija	Vraća ukupan broj elemenata koje kolekcija trenutno sadrži
LIMIT	Funkcija	Vraća ukupan broj elemenata koje kolekcija može imati. Kod polja varijabilne duljine vraća fiksnu veličinu koja je definirana.
FIRST	Funkcija	Vraća indeks prvog elementa kolekcije. U slučaju da se prvi element ugniježđene tablice izbriše, funkcija FIRST vraća vrijednost veću od 1.
LAST	Funkcija	Vraća indeks zadnjeg elementa kolekcije. U slučaju da se elementi koji se nalaze u sredini ugniježđene tablice izbrišu, funkcija LAST vraća vrijednost koja je veća od one koju vrati funkcija COUNT
PRIOR	Funkcija	Vraća indeks neposredno prije navedenog indeksa
NEXT	Funkcija	Vraća indeks neposredno nakon navedenog indeksa
EXTEND	Procedura	Povećava veličinu kolekcije kod ugniježđenih tablica i polja varijabilne duljine

TRIM	Procedura	Uklanja jedan ili određeni broj elemenata s kraja kolekcije(najviše definirani indeks) kod ugniježđenih tablica i polja varijabilne duljine. Može obrisati vrijednost polja
DELETE	Procedura	Briše jedan ili sve elemente iz kolekcije, elemente u nekom rasponu ili određeni element. Brišu se samo vrijednosti polja.

Tablica 6 Prikaz metoda kolekcija

7.2.4 Višerazinske kolekcije

PL/SQL omogućava kreiranje višerazinskih kolekcija, odnosno kolekcija koje su ugniježđene jedna unutar druge, odnosno tipovi višerazinskih kolekcija su tipovi kolekcija kod kojih se za tip podataka upotrebljava druga kolekcija. Moguće kombinacije višerazinskih tipova podataka su:

- Ugniježđena tablica koja za tip ima ugniježđenu tablicu
- Ugniježđena tablica koja za tip ima polje varijabilne duljine
- Polje varijabilne duljine koje za tip ima ugniježđenu tablicu
- Polje varijabilne duljine koje za tip ima polje varijabilne duljine

8. Obrada grešaka

Unutar PL/SQL-a svaka greška ili upozorenje bilo koje vrste naziva se iznimkom (eng. *Exception*) te se one ne bi smjele događati unutar programa prilikom izvođenja bloka koda. Postoje dvije vrste PL/SQL grešaka, a to su greške kod prevođenja programa (eng. *Compile-time errors*, poznatije i kao sintaksne greške[19]) te greške kod izvođenja programa (eng. *Run-time errors*, poznatije i kao semantičke greške). Greške kod prevođenja pojavljuju se u anonimnim i imenovanim blokovima, odnosno u funkcijama, procedurama, paketima te ih je lakše otkriti zato što Oracle stroj odmah šalje upozorenje. Greške kod izvođenja uzrokovane su kada je logika programa pogrešna te su zbog toga složenije i teže ih je riješiti zato što se javljaju samo povremeno te su stoga dizajnirane iznimke za rješavanje grešaka prilikom izvođenja[20]. Deklaracija iznimaka odvija se u deklarativnom dijelu bloka, pozivanje iznimaka izvršava se u izvršnom dijelu bloka, a obrađuju se u dijelu bloka za obradu iznimaka. Osnovna sintaksa za obradu iznimkama je:

```
DECLARE
        ...
BEGIN
        izvršne_naredbe;
EXCEPTION
        WHEN ime_iznimke THEN
                naredbe_koje_se_izvrše_kod_pojave _pogreške
END;
```

Pojavom iznimke[21], izvođenje programa se zaustavlja, a kontrola se premješta u dio bloka za obradu iznimaka[22] ili u potprogram. Korištenje rukovatelja iznimaka olakšava pisanje i razumijevanje programa te se smanjuje mogućnost pojave neobrađenih iznimaka. U suprotnome trebala bi se provjeriti svaka mogućnost pojave iznimke (velika vjerojatnost da se predvidi moguća greška), te ukoliko se pojavi potrebno ju je i obraditi.

Iznimke se sastoje od četiri atributa:

- Ime – daje kratak opis problema; NO_DATA_FOUND

[19] Prilikom pisanja programa zaboravi se napisati zarez, zagrada, točka ili identifikator i tako dalje.
[20] Drugim riječima greške kod izvođenja nazivaju se iznimkama.
[21] Oracle stroj prvi otkrije iznimku ukoliko se pojavi.
[22] Drugi naziv za dio bloka za obradu iznimaka je rukovatelj iznimkama (eng. *Exception handler*) koji poduzima odgovarajuće radnje protiv iznimaka.

- Tip – identificira mjesto greške (to može biti PL/SQL jezik, Oracle kernel i tako dalje); ORA
- Šifra iznimke; -1403
- Poruka o grešci – pruža dodatne informacije o iznimci; no data found

Primjer koda za model krojačnice:

```
DECLARE
        postanskiBroj mjesto.postanski_broj%TYPE;
        nazivMjesta mjesto.naziv_mjesta%TYPE;
BEGIN
        SELECT  postanski_broj, naziv_mjesta
        INTO postanskiBroj, nazivMjesta
        FROM mjesto
        WHERE postanski_broj = 43000;
  DBMS_OUTPUT.PUT_LINE ('Poštansk broj mjesta: '||  postanskiBroj);
  DBMS_OUTPUT.PUT_LINE ('Naziv mjesta: '||  nazivMjesta);

EXCEPTION
        WHEN NO_DATA_FOUND THEN
                DBMS_OUTPUT.PUT_LINE('Podaci o mjestu ne postoje!!');
        WHEN OTHERS THEN
                DBMS_OUTPUT.PUT_LINE('Greška!!');
END;
/
```

Rezultat:

Podaci o mjestu ne postoje!!

PL/SQL procedure successfully completed.

Iznimke se dijele na dvije kategorije:

- Ugrađene iznimke (eng. *Built-in Exceptions*)
- Korisnički definirane iznimke (eng. *User-Defined Exceptions*)

8.1 Ugrađene iznimke

Kada se pojavi greška unutar programa koja je uzrokovana kršenjem pravila, ugrađene se iznimke pozivaju implicitno (automatski), odnosno kontrola se prosljeđuje dijelu bloka za obradu iznimaka unutar kojeg se izvršavaju dane naredbe za obradu grešaka. Blok završava nakon što se izvrše naredbe unutar bloka za obradu iznimaka te se kontrola ne vraća u izvršni dio bloka. Unutar globalnog Oracleovog paketa STANDARD nalaze se predefinirane iznimke s unaprijed definiranim nazivima i odmah su dostupne unutar programa (nije ih potrebno deklarirati kao korisnički definirane iznimke). Osim imena sadrže i broj greške. Tablica 7 prikazuje nekoliko primjera ugrađenih iznimaka.

Naziv iznimke	**Broj greške**	**Opis**
ACCESS_INTO_NULL	ORA-06530	Automatski se dodjeljuje vrijednost NULL objektu
CASE_NOT_FOUND	ORA-06592	Nijedna WHEN klauzula u CASE naredbi nije zadovoljena i nije navedena ELSE klauzula
COLLECTION_IS_NULL	ORA-06531	Korištenje metoda kolekcija (osim EXISTS) ili pristupanje atributima kolekcije na ne inicijaliziranim kolekcijama
CURSOR_ALREADY_OPEN	ORA-06511	Kod otvaranje kursora koji je već otvoren
DUP_VAL_ON_INDEX	ORA-00001	Kada se duple vrijednosti pokušavaju pohraniti u stupac s jedinstvenim indeksom
INVALID_CURSOR	ORA-01001	Kada se pokušaju izvršiti nedopuštene operacije nad kursorom poput

		zatvaranja neotvorenog kursora
INVALID_NUMBER	ORA-01722	Kod pretvorbe niza znakova u broj ali neuspješno zato što niz ne predstavlja valjani broj
NO_DATA_FOUND	ORA-01403	Kada SELECT naredba koja sadrži INTO klauzulu ne dohvati nijedan redak
ROWTYPE_MISMATCH	ORA-06504	Kada kursor dohvaća vrijednosti u varijabli koja ima nekompatibilan tip podataka
STORAGE_ERROR	ORA-06500	Pojavljuje se kada PL/SQL ostaje bez memorije ili kada je memorija oštećena
TOO_MANY_ROWS	ORA-01422	Kada SELECT naredba s INTO klauzulom vrati više od jednog retka
VALUE_ERROR	ORA-06502	Kada se pojavi aritmetička greška ili greška ograničenja veličine (na primjer kada se dodijeli vrijednost varijable koja je veća od veličine varijable)
ZERO_DIVIDE	ORA-01476	Pojavljuje se kada se broj pokuša dijeliti s nulom

Tablica 7 Prikaz ugrađenih iznimaka

8.2 Korisnički definirane iznimke

Korisnik može deklarirati sam svoje iznimke u deklarativnom dijelu anonimnog bloka, potprograma ili paketa prema potrebama programa. Deklaracija iznimke sastoji se od njena naziva nakon čega slijedi ključna riječ EXCEPTION:

```
DECLARE
        naziv_iznimke EXCEPTION;
```

Eksplicitno se pozivaju korištenjem RAISE naredbi ili korištenjem ugrađene procedure RAISE_APPLICATION_ERROR. Drugim riječima programu treba navesti pod kojim se okolnostima iznimka poziva. Sintaksa za korisnički definiranu iznimku je:

```
DECLARE
        naziv_iznimke EXCEPTION;
BEGIN
        IF uvjet THEN
                RAISE naziv_iznimke;
        ELSE
                …
        END IF;
EXCEPTION
        WHEN naziv_iznimke THEN
        naredbe;
END;
```

IF-THEN-ELSE naredba (može se koristiti svaki oblik IF naredbe) koristi se za utvrđivanje uvjeta pod kojim se poziva korisnički definirana iznimka. Ako je vrijednost *uvjeta* TRUE pozvati će se korisnički definirana iznimka, a ukoliko je vrijednost FALSE, program nastavlja s radom na uobičajen način.

RAISE naredba može poprimiti jedan od tri navedena oblika:

- RAISE; - ne zahtjeva naziv iznimke, može se koristiti samo unutar WHEN klauzule unutar bloka za obradu iznimaka, odnosno kada se ta ista iznimka ponovno poziva ili propagira unutar bloka za obradu iznimaka
- RAISE *naziv_iznimke*; - koristi se kod pozivanja iznimke koja je deklarirana u trenutnom ili u vanjskom bloku koji sadrži taj blok ili kod pozivanja ugrađenih iznimaka definiranih unutar STANDARD paketa

- RAISE *naziv_paketa.naziv_iznimke*; - potrebno je koristiti naziv paketa. U slučaju da je iznimka deklarirana unutar nekog drugog paketa (nije korišten STANDARD paket) i iznimka se poziva van tog paketa, potrebno je dodati ime paketa na iznimku unutar RAISE naredbe

Primjer koda za model krojačnice:

```
DECLARE
   var_spol VARCHAR2(8);
   poruka  VARCHAR2(300);
   iznimka_krivi_spol EXCEPTION;
BEGIN
   SELECT spol_klijenta INTO var_spol FROM klijent WHERE sifra_klijenta = 357;
   DBMS_OUTPUT.PUT_LINE('Spol klijenta: ' || var_spol);
   IF var_spol = 'Muško' THEN
                poruka := 'muško.';
    ELSE
                RAISE iznimka_krivi_spol;
   END IF;
   DBMS_OUTPUT.PUT_LINE('Klijent je ' || poruka);
EXCEPTION
   WHEN iznimka_krivi_spol THEN
     DBMS_OUTPUT.PUT_LINE('Dogodila se greška!!!');
END;
/
```

Rezultat: Ispis poruke kada je spol klijenta *Muško* te se iznimka ne poziva nego se ispiše pripadajuća poruka.

```
Spol klijenta: Muško
Klijent je muško.

PL/SQL procedure successfully completed.
```

Ispis poruke kada je spol klijenta *Žensko*, poziva se iznimka i ispiše se pripadajuća poruka.

```
Spol klijenta: Žensko
Dogodila se pogreška!!!

PL/SQL procedure successfully completed.
```

Ugrađena procedura **RAISE_APPLICATION_ERROR** definirana je unutar DBMS_STANDARD paketa te omogućuje korisniku definiranje vlastitih poruka o grešci iz pohranjenih potprograma ili metoda (svakoj poruci pridružuje se broj greške). Sintaksa je:

RAISE_APPLICATION_ERROR(*broj_greske, poruka*[, {TRUE|FALSE}]);

gdje *broj_greške* predstavlja negativni cijeli broj u rasponu od -20 000 do -20 999, *poruka* je znakovni niz (string) koji predstavlja poruku o grešci do dužine 2048 bajta. Zadnji parametar su Boolean vrijednosti koje nisu obavezne. Ako je vrijednost parametra FALSE (definirana vrijednost) greška zamjenjuje sve prethodne greške. A ako je vrijednost parametra TRUE, greška se dodaje na stog prethodnih grešaka.

Primjer koda za model krojačnice:

```
DECLARE
        var_spol VARCHAR2(10);
        poruka  VARCHAR2(250);
BEGIN
        SELECT spol_klijenta
         INTO var_spol
         FROM klijent
         WHERE sifra_klijenta = 7;
        IF var_spol = 'Muško' THEN
                poruka := 'muško';
        ELSE
                RAISE_APPLICATION_ERROR(-20101, 'Dogodila se greška!!');
   END IF;
   DBMS_OUTPUT.PUT_LINE('Klijent je: ' || poruka);
END;
/
```

Rezultat:

Error report -
ORA-20101: Dogodila se greška!!
ORA-06512: at line 9

Nakon što je iznimka pozvana, PL/SQL blok prestaje s izvođenjem te prosljeđuje kontrolu bloku za obradu iznimaka (više nije moguć povratak u izvršni dio bloka). U slučaju da taj blok izostane, iznimka se prosljeđuje vanjskom bloku. Kada se pojavi greška koja je povezana s iznimkom, a iznimka je prethodno pozvana, potrebno je napisati kod za hvatanje i obradu iznimaka (eng. *Exception handling*) koji se potom izvršava. Kod za hvatanje i obradu

iznimkama piše se nakon svih izvršnih naredbi ali prije END bloka. Svaki PL/SQL blok može imati dio za obradu iznimaka koji se sastoji od jedne ili više iznimki (strukturirane su poput CASE naredbi). Sintaksa je:

```
EXCEPTION
        WHEN naziv_iznimke1 THEN
                naredba1;
        WHEN naziv_iznimke2 OR naziv_iznimke3 THEN
                naredba2;
        WHEN OTHERS THEN
                naredba3;
END;
```

Sintaksa se sastoji od WHEN klauzule i naredbi koje se izvrše kada je iznimka pozvana. Ako se pozove *naziv_iznimke1* tada se izvodi *naredba1*. Ako su pozvane *naziv_iznimke2* ili *naziv_iznimke3* tada se izvodi *naredba2.* Ako je pozvana bilo koja druga iznimka, tada se izvodi *naredba3.* Prednost korištenja blokova za obradu iznimaka je ta da omogućuju lakše pisanje koda na način da su lakši za razumijevanje te sprečavaju pojavu mogućih grešaka koje su uzrokovane ljudskim faktorom.

8.2.1 WHEN OTHERS klauzula

Može se dogoditi da naziv iznimke ne odgovara nazivu iznimke unutar WHEN klauzule, tada se izvršavaju naredbe koje su povezane s **WHEN OTHERS** klauzulom. Samo jedan blok za obradu iznimaka može uhvatiti određenu grešku. Nakon što su naredbe izvršene unutar tog bloka, kontrola se odmah prosljeđuje izvan tog bloka. Dobra je praksa da se OTHERS klauzula koristi na početku programa (vanjski blok) kako bi se izbjegle moguće neotkrivene greške. Korištenje WHEN OTHERS klauzule nije obavezno. Unutar OTHER klauzule postoji nekoliko funkcija:

Naziv funkcije	Opis
SQLCODE	Vraća šifru greške koja je posljednja pozvana unutar bloka. Vraća 0 ukoliko greška ne postoji ili ako se greška poziva izvan dijela bloka za obradu iznimaka

SQLERRM	Vraća poruku o grešci za određenu šifru greške. Ako se SQLERRM-u ne proslijedi šifra greške vraća poruku o grešci koja je povezana s vrijednošću koju vraća SQLCODE. Ako SQLCODE iznosi 0, SQLERRM vraća poruku: *ORA-0000: normal, successful completion*. Ako SQLCODE iznosi 1, SQLERRM vraća poruku: *User-Defined Exception*
DBMS_UTILITY.FORMAT_ERROR_STACK	Vraća poruku koja je povezana s trenutnom greškom (vraća iste informacije kao i SQLERRM). Ograničen je na 2 000 bajtova.
DBMS_UTILITY.FORMAT_ERROR_BACKTRACE	Vraća stog grešaka na mjesto gdje je greška izvorno pozvana

Tablica 8 Prikaz funkcija WHEN OTHERS klauzule

Primjer koda za model krojačnice:

```
DECLARE
        unos_sifre NUMBER := &unos;
        ime VARCHAR(25);
        prezime VARCHAR(25);
        broj_greške NUMBER;
        poruka_o_grešci VARCHAR2(200);

BEGIN
        SELECT ime_klijenta, prezime_klijenta
         INTO ime, prezime
        FROM klijent
        WHERE sifra_klijenta = unos_sifre;
EXCEPTION
        WHEN OTHERS THEN
                broj_greške := SQLCODE;
                poruka_o_grešci := SUBSTR(SQLERRM, 1, 200);
```

```
            DBMS_OUTPUT.PUT_LINE ('Error code: ' || broj_greške);
            DBMS_OUTPUT.PUT_LINE ('Error message: ' || poruka_o_grešci);
END;
/
```

Rezultat:

Error message: ORA-01403: no data found

PL/SQL procedure successfully completed.

U primjeru se traži unos šifre klijenta. Ako se unese šifra koja nije spremljena u bazu poziva se iznimka NO_DATA_FOUND.

8.2.2 EXCEPTION_INIT Pragma

Isto kao i procedura RAISE_APPLICATION_ERROR, EXCEPTION_INIT pragma omogućuje da korisnik samostalno definira svoje poruke o grešci i broj greške. Odnosno, ukoliko postoji Oracle greška koja ima samo broj greške, a naziv je izostavljen, nije moguće napisati kod koji bi uhvatio i obradio tu grešku. U tom slučaju koristi se *pragma*[23]. U konačnici EXCEPTION_INIT pragma omogućuje da se broj greške poveže sa imenom korisnički definirane greške. Sintaksa je:

PRAGMA EXCEPTION_INIT (*naziv_iznimke*, *broj_greške*);

gdje *naziv_iznimke* označava naziv iznimke deklarirane prije pragme, dok *broj_greške* označava željeni broj greške koji je povezan s tim nazivom iznimke.

Primjer koda za model krojačnice:

```
DECLARE
        cijena_popravka_previsoka EXCEPTION;
        PRAGMA EXCEPTION_INIT(cijena_popravka_previsoka, - 20111);
        max_cijena_usluge cjenik_usluga.cijena_usluge%TYPE;
        sifra_sluge_unos cjenik_usluga.sifra_usluge%TYPE := &sifra_usluge;
        cijena_usluge_unos cjenik_usluga.cijena_usluge%TYPE := &cijena_usluge;
BEGIN
        SELECT MAX(cijena_usluge)
        INTO max_cijena_usluge
```

[23] Pragma je posebna uputa za PL/SQL prevoditelj koja se procesuira za vrijeme kompajliranja. Nalazi se u deklarativnom dijelu bloka nakon deklariranja korisnički definirane iznimke [1].

```
        FROM cjenik_usluga;

        IF cijena_usluge_unos > max_cijena_usluge THEN
            RAISE_APPLICATION_ERROR(-20111,'Cijena popravka previsoka!!');
        END IF;

        UPDATE cjenik_usluga
        SET cijena_usluge = cijena_usluge_unos
        WHERE sifra_usluge = sifra_sluge_unos;

        COMMIT;
END;
/
```

Rezultat:

	SIFRA_USLUGE	NAZIV_USLUGE	CIJENA_USLUGE
1	1	Šivanje	45
2	2	Krojenje	150
3	3	Strojno skraćivanje hlača	45
4	4	Skraćivanje kaputa	130
5	5	Sužavanje haljine	90
6	6	Zamjena podstave na suknji	40
7	7	Produljivanje rukava	80
8	8	Sužavanje suknji	70

Slika 21 Prikaz tablice CJENIK USLUGA prije ažuriranja šifre usluge pod brojem 4

	SIFRA_USLUGE	NAZIV_USLUGE	CIJENA_USLUGE
1	1	Šivanje	45
2	2	Krojenje	150
3	3	Strojno skraćivanje hlača	45
4	4	Skraćivanje kaputa	120
5	5	Sužavanje haljine	90
6	6	Zamjena podstave na suknji	40
7	7	Produljivanje rukava	80
8	8	Sužavanje suknji	70

Slika 22 Prikaz tablice CJENIK USLUGE nakon ažuriranja šifre usluge pod brojem 4

Primjer prikazuje kako treba deklarirati korisničku definiranu iznimku *cijena_popravka_previsoka* kojoj se pridružuje broj greške - 20111. U drugom koraku odabire se maksimalna cijena usluge iz tablice CJENIK USLUGA pomoću funkcije MAX() i dobivena vrijednost sprema se u varijablu *max_cijena_usluge*. Nakon toga vrši se provjera unosa maksimalne cijene usluge te ukoliko je cijena veća od maksimalne cijene iz tablice,

poziva se RAISE_APPLICATION_ERROR. U zadnjem koraku ažurira se *cijena_usluge* čiju je šifru unio korisnik s novim ograničenjem. Ukoliko korisnik unese veću cijenu usluge od maksimalne prikazati će se poruka o grešci:

Error report -
ORA-20111: Cijena popravka previsoka!!
ORA-06512: at line 13

8.2.3 Propagacija iznimaka

U slučajevima pozivanja iznimaka u kojima je izostavljen dio bloka za obradu iznimaka ili se iznimka poziva iz različitog dijela bloka, PL/SQL propagira[24] iznimku. Postoje tri situacije u kojima se može dogoditi propagacija. Ukoliko se dogodi propagacija iznimke, koja je pozvana u **izvršnom dijelu bloka** te ukoliko postoji dio bloka za obradu iznimaka, blok se izvršava te se kontrola prosljeđuje aplikacijskom okruženju ili vanjskom bloku. Ako se dogodi da niti u jednom bloku nije bio dio bloka za obradu iznimaka, iznimka se propagira na vanjski blok. Postupak se ponavlja u svakom sljedećem vanjskom bloku sve dok ne ponestane blokova u kojima je moguće pozvati iznimku. Ukoliko se ponovno dogodi da niti u jednom bloku nije bilo dijela bloka za obradu iznimaka, zaustavlja se izvršenje bloka te se kontrola vraća se u aplikacijsko okruženje, što će na kraju odrediti ishod. Druga situacije je propagacija iznimke koja je pozvana u **deklarativnom dijelu bloka** u kojoj se iznimka direktno propagira na vanjski blok. Ukoliko se dogodi da nema vanjskog bloka, izvršenje bloka se zaustavlja te se kontrola prosljeđuje u aplikacijsko okruženje. Potrebno je naglasiti da se dio za obradu iznimaka ne smije nalaziti u deklarativnom dijelu bloka nego u vanjskom bloku [7] [9] [36].

Primjer:

```
BEGIN
  DECLARE
        var_string VARCHAR2 (5) := 'Ovo je proba';
  BEGIN
       DBMS_OUTPUT.put_line (var_string);
  EXCEPTION
    WHEN VALUE_ERROR
    THEN
      DBMS_OUTPUT.PUT_LINE ('Iznimka je obrađena');
```

[24] Propagacija iznimaka su pravila koja određuju kako će se iznimke pozivati u tim situacijama.

```
  END;
EXCEPTION
  WHEN VALUE_ERROR
  THEN
    DBMS_OUTPUT.PUT_LINE ('Kontrola se prosljeđuje vanjskom bloku');
END;
/
```

Rezultat:

Kontrola se prosljeđuje vanjskom bloku

PL/SQL procedure successfully completed.

U zadnjoj situacija dolazi do propagacije iznimke koja je pozvana u **dijelu za obradu iznimaka,** tada se iznimka direktno propagira na vanjski blok. Istovremeno ne može biti pozvano više od jedne iznimke nego jedna po jedna. Ukoliko i tu izostane vanjski blok, izvršenje bloka se zaustavlja te se kontrola prosljeđuje u aplikacijsko okruženje. Dio za obradu iznimaka mora se nalaziti u vanjskom bloku.

Primjer:

```
--vanjski blok
DECLARE
        iznimka1 EXCEPTION;
        iznimka2 EXCEPTION;
BEGIN
-- unutarnji blok
        BEGIN
                RAISE iznimka1;
        EXCEPTION
                WHEN iznimka1 THEN
                        RAISE iznimka2;
                WHEN iznimka2 THEN
                        DBMS_OUTPUT.PUT_LINE ('Dogodila se greška u
                unutarnjem bloku');
        END;
EXCEPTION
        WHEN iznimka2 THEN
                DBMS_OUTPUT.PUT_LINE ('Dogodila se greška unutar programa');
END;
/
```

Rezultat:

Dogodila se greška unutar programa

PL/SQL procedure successfully completed.

Za primjer pozivanja iznimke u dijelu za obradu iznimaka deklarirale su se dvije iznimke *iznimka1* i *iznimka2*. *iznimka1* poziva se u unutarnjem bloku pomoću RAISE naredbe. U dijelu za obradu iznimaka, *iznimka1* pokušava pozvati *iznimku2* u unutarnji blok te se kontrola prosljeđuje vanjskom bloku. Razlog tomu je što se može pozvati samo jedna iznimka unutar dijela za obradu iznimaka. Nakon obrade prve iznimke, poziva se druga iznimka.

8.2.4 Ponovno pozivanje iznimaka

Iznimke se ponovno pozivaju korištenjem RAISE naredbe ukoliko se pojavi greška u unutarnjem bloku i ako se iznimka želi obraditi u vanjskom bloku. Za ponovno pozivanje iznimke nije potrebno navesti ime iznimke.

Primjer koda za model krojačnice:

```
DECLARE
        cijena_popravka_previsoka EXCEPTION;
        PRAGMA EXCEPTION_INIT( cijena_popravka_previsoka, -20001);
        max_cijena_usluge cjenik_usluga.cijena_usluge%TYPE;
        sifra_sluge_unos cjenik_usluga.sifra_usluge%TYPE := &sifra_usluge;
        cijena_usluge_unos cjenik_usluga.cijena_usluge%TYPE := &cijena_usluge;
BEGIN
        BEGIN
                SELECT MAX(cijena_usluge)
                INTO max_cijena_usluge
                FROM cjenik_usluga;

                IF cijena_usluge_unos > max_cijena_usluge THEN
                        RAISE cijena_popravka_previsoka;
                END IF;
                EXCEPTION
                        WHEN cijena_popravka_previsoka THEN
                                DBMS_OUTPUT.PUT_LINE('Cijena popravka od ' ||
cijena_usluge_unos || ' kuna je previsoka!!');
                                RAISE; -- ponovno pozivanje iznimke
        END;
```

```
EXCEPTION
        WHEN cijena_popravka_previsoka THEN
                SELECT AVG(cijena_usluge)
                INTO cijena_usluge_unos
                FROM cjenik_usluga;

                DBMS_OUTPUT.PUT_LINE('Nova cijena popravka iznosi: ' ||
cijena_usluge_unos || ' kuna');

                UPDATE cjenik_usluga
                SET cijena_usluge = cijena_usluge_unos
                WHERE sifra_usluge = sifra_sluge_unos;
                COMMIT;
END;
/
```

Rezultat:

	SIFRA_USLUGE	NAZIV_USLUGE	CIJENA_USLUGE
1	1	Šivanje	45
2	2	Krojenje	150
3	3	Strojno skraćivanje hlača	45
4	4	Skraćivanje kaputa	130
5	5	Sužavanje haljine	90
6	6	Zamjena podstave na suknji	40
7	7	Produljivanje rukava	80
8	8	Sužavanje suknji	70

Slika 23 Prikaz tablice CJENIK USLUGA prije ažuriranja šifre usluge pod brojem 6

	SIFRA_USLUGE	NAZIV_USLUGE	CIJENA_USLUGE
1	1	Šivanje	45
2	2	Krojenje	150
3	3	Strojno skraćivanje hlača	45
4	4	Skraćivanje kaputa	130
5	5	Sužavanje haljine	90
6	6	Zamjena podstave na suknji	81,25
7	7	Produljivanje rukava	80
8	8	Sužavanje suknji	70

Slika 24 Prikaz tablice CJENIK USLUGA poslije ažuriranja šifre usluge pod brojem 6

Primjer prikazuje ponovno pozivanje iznimke nad tablicom CJENIK USLUGA. Odabire se maksimalna cijena usluge pomoću funkcije MAX() i dobivena vrijednost sprema se u

varijablu *max_cijena_usluge.* Nakon toga vrši se provjera unosa maksimalne cijene usluge i ukoliko je cijena veća od maksimalne cijene iz tablice, tada se poziva iznimka *cijena_popravka_previsoka*. Ispisuje se poruka i iznimka se ponovno poziva u dijelu za obradu iznimke u unutarnjem bloku. U vanjskom bloku pomoću funkcije AVG() dodjeljuje se nova prosječna vrijednost varijabli *cijena_usluge_unos* te se ispisuje pripadajuća poruka:

Cijena popravka od 1000 kuna je previsoka!!
Nova cijena popravka iznosi: 81,25 kuna

PL/SQL procedure successfully completed.

Na kraju tablica se ažurira s novom vrijednošću za cijene usluge pod šifrom 6.

9. Potprogrami

Prethodno je spomenuto da se blokovi dijele na anonimne i imenovane blokove. U radu su do sada prikazani primjeri samo za anonimne blokove. Temeljna razlika između anonimnih i imenovanih blokova je ta da se anonimni blokovi ne spremaju u bazu podataka i ne mogu se pozvati direktno iz drugih blokova te počinju s ključnom riječju DECLARE ili BEGIN. Imenovani blokovi su ustvari potprogrami koji se mogu pozvati više puta i koji mogu prihvatiti parametre. Unutar PL/SQL-a nalaze se dvije vrste potprograma [9] [37] [38]:

- **Procedure** koje ne vraćaju vrijednosti izravno nego izvršavaju određene obrade nad podacima
- **Funkcije** koje se koriste za izračunavanje i vraćanje vrijednosti pomoću RETURN klauzule

Na početku potprograma nalazi se zaglavlje unutar kojeg se navodi naziv i parametri potprograma koji nisu obavezni. Potprogrami se sastoje od:

- **Deklarativnog dijela** koji nije obavezan te unutar kojeg se nalaze deklaracije tipova, kursora, konstanti, varijabli, iznimaka i ugniježđenih potprograma. Na početku se izostavlja ključna riječ DECLARE
- **Izvršnog dijela** koji je obavezan te unutar kojeg se nalaze naredbe kojima se dodjeljuju vrijednosti i koje kontroliraju izvršavanje i manipulaciju podataka
- **Dijela za obradu iznimaka** koji nije obavezan i koji sadrži kod za obradu grešaka prilikom izvođenja

Potprogrami mogu biti kreirani:

- Na razini sheme (eng. *Standalone subprogram*) pomoću naredbe CREATE PROCEDURE ili CREATE FUNCTION. Spremaju se u bazu podataka te se mogu obrisati naredbom DROP PROCEDURE ili DROP FUNCTION
- Unutar paketa (eng. *Packaged subprogram*) koji se pohranjuju u bazu podataka i mogu se izbrisati samo u slučaju kada je paket obrisan naredbom DROP PACKAGE
- Unutar PL/SQL bloka (eng. *Nested subprogram*) koji se može deklarirati i definirati u isto vrijeme ili se prvotno deklarira, a zatim se kasnije definira u istom bloku. Pohranjuje se u bazu podataka jedino u slučaju ako je ugniježđen na razini sheme ili unutar paketa

Neke od prednosti korištenja potprograma su:

- Modularnost (eng. *Modularity*) - omogućava da se veliki dijelovi koda podijele na manje module koje mogu pozvati drugi moduli
- Održavanje (eng. *Maintainability*) – mogućnost promjene potprograma bez da se mijenja mjesto pozivanja
- Ponovno korištenje (eng. *Reusability*) – korištenje potprograma više puta od strane drugih aplikacija
- Korištenje paketa (eng. *Packageability*) – grupiranje potprograma u pakete

9.1 Procedure

Procedura je imenovani blok, odnosno potprogram koji izvodni određene zadatke. Bitna razlika između procedure i funkcije je ta da procedura ne vraća vrijednosti. Procedura se sastoji od naziva procedure, parametara, lokalnih varijabli i BEGIN-END bloka. Procedura se sprema u bazu podataka kao objekt baze podataka. Sintaksa za izradu procedura je:

[CREATE [OR REPLACE]] PROCEDURE *naziv_procedure*
    [(*naziv_parametra* [IN | OUT | IN OUT] *podatkovni_tip* [, ...])]
{IS | AS}
    [*lokalne deklaracije*]
BEGIN
    izvršne_naredbe
[EXCEPTION
    obrade_grešaka]
END [*naziv_procedure*];

gdje *naziv_procedure* označava naziv same procedure. [OR REPLACE] opcija omogućava izmjenu postojeće procedure ali nije obavezna za korištenje. U slučaju da nije korištena u zaglavlju procedure i da bi se mogle napraviti potrebne izmjene unutar koda, potrebno ju je najprije obrisati pomoću naredbe DROP PROCEDURE *naziv_procedure* i ponovno kreirati. *naziv_parametra* sastoji se od izborne liste parametara koja sadrži ime i tip pridruženog parametra koji se definiraju kako bi se informacije proslijedile proceduri i poslale izvan procedure nazad pozivnom programu. *lokalne deklaracije* omogućavaju deklariranje lokalnih identifikatora funkcije. Ako kod ne sadrži niti jednu deklaraciju, između ključnih riječi IS i

BEGIN neće se nalaziti niti jedna naredba. *izvršne_naredbe* su naredbe koje se izvršavaju kada su pozvane. Potrebno je imati barem jednu izvršnu naredbu nakon ključne riječi BEGIN i prije ključne riječi END ili EXCEPTION. *obrade_grešaka* navode dio za obradu grešaka.

Primjer koda za model krojačnice:

```
CREATE OR REPLACE PROCEDURE prikazi_podatke_klijenta( klijent_sifra
NUMBER)
IS
        podaci_klijent klijent%ROWTYPE;
BEGIN
SELECT *
INTO podaci_klijent
FROM klijent
WHERE sifra_klijenta = klijent_sifra ;

DBMS_OUTPUT.PUT_LINE ('Ime: '|| podaci_klijent.ime_klijenta
                    ||chr(13)
                    ||'Prezime: '
                    ||podaci_klijent.prezime_klijenta
                    ||chr(13)
                    ||'Spol: '
                    ||podaci_klijent.spol_klijenta
                    ||chr(13)
                    ||'Puta popravljao: '
                    ||podaci_klijent.puta_popravljao
                    ||chr(13)
                    ||'Ukupno potrošio na šivanje: '
                    ||podaci_klijent.ukupno_potrosio_sivanje
                    ||' kn.');
EXCEPTION
        WHEN OTHERS THEN
                DBMS_OUTPUT.PUT_LINE( SQLERRM);
END;
```

Pozivanje procedure:

```
BEGIN
prikazi_podatke_klijenta  (357);
END;
```

Rezultat:

Ime: Marijan

Prezime: Horvat
Spol: Muško
Puta popravljao: 3
Ukupno potrošio na šivanje: 821 kn.

PL/SQL procedure successfully completed.

Pozivanjem procedure iz primjera ispisuju se neke od bitnijih informacija o klijentu. Procedura se unutar programa SQL Developer sprema u odjeljku *Procedures.* Za pozivanje procedure koristi se sintaksa EXECUTE *ime_procedure*(*argumenti*).

9.2 Parametri potprograma

Parametri potprograma koriste se za prijenos vrijednosti[25] između potprograma i programa. Postoje dvije vrste parametara, a to su:

- Formalni parametri (eng. *Formal Parameters*)
- Stvarni parametri (eng. *Actual Parameters*)

Formalni parametri su imena koja su deklarirana u listi parametara u zaglavlju potprograma. Može ih se smatrati rezerviranim mjestima za vrijednosti pravih parametara koja se prosljeđuju potprogramu. Stvarni parametri su vrijednosti ili izrazi u listi parametara koji se postavljaju kod pozivanja potprograma. Postoje tri vrste parametara [38] [39] [7] [1]:

Naziv parametra	**Uporaba**	**Formalni parametar**	**Stvarni parametar**	**Opis**
IN	Samo za čitanje (eng. *Read-only*)	Unutar potprograma ponaša se kao konstanta te mu se ne može dodijeliti vrijednost	Može biti konstanta, inicijalizirana varijabla, literal ili izraz	Omogućuje prosljeđivanje vrijednosti potprogramu. IN je zadani parametar ukoliko se ne navede druga vrsta parametra.

[25] To su vrijednosti koje se dobiju obradom ili se vraćaju nakon obrade procedure.

OUT	Samo za pisanje (eng. *Write-only*)	Unutar potprograma ponaša se poput varijable koja nije inicijalizirana te mu se vrijednosti može promijeniti.	Mora biti varijabla.	Vraća vrijednost pozivatelju programa.
IN OUT	Za čitanje i pisanje (eng. *Read/Write*)	Ponaša se poput inicijalizirane varijable te mu se mora dodijeliti vrijednost	Mora biti varijabla	Prosljeđuje početnu vrijednost potprogramu i vraća izmijenjenu vrijednost pozivatelju potprograma. IN OUT parametar ne može biti konstanta, literal ili izraz.

Tablica 9 Prikaz vrsti parametara potprograma

Primjer koda za model krojačnice:

```
CREATE OR REPLACE PROCEDURE ispisi_klijenta
        (sifraKlijenta IN NUMBER,
        imeKlijenta OUT VARCHAR2,
        prezimeKlijenta OUT VARCHAR2)
AS
BEGIN
        SELECT ime_klijenta, prezime_klijenta
        INTO imeKlijenta, prezimeKlijenta
        FROM klijent
        WHERE sifra_klijenta = sifraKlijenta;
EXCEPTION
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE('Greška!! Ne postoji klijent pod šifrom: '||sifraKlijenta);
END ispisi_klijenta;
```

Pozivanje procedure:

```
DECLARE
        var_ime_klijenta klijent.ime_klijenta%TYPE;
```

```
        var_prezime_klijenta klijent.prezime_klijenta%TYPE;
BEGIN
        ispisi_klijenta
                (123, var_ime_klijenta, var_prezime_klijenta);
DBMS_OUTPUT.PUT_LINE ('Pod šifrom 123 nalazi se klijent: '
                ||var_ime_klijenta
                ||' '
                || var_prezime_klijenta);
END;
```

Rezultat:

Pod šifrom 123 nalazi se klijent: Danijela Glogoški

PL/SQL procedure successfully completed.

Procedura *ispisi_klijenta* uzima za parametar *sifraKlijenta.* Parametri *imeKlijenta* i *prezimeKlijenta* su tipa OUT. Procedura je jednostavna SELECT naredba koja vraća *ime_klijneta* i *prezime_klijenta* iz tablice KLIJENT gdje je *sifra_klijenta* jednaka vrijednosti *sifraKlijenta* i ta varijabla je jedini parametar tipa IN unutar procedure. Za pozivanje procedure, vrijednost se mora proslijediti u parametar *sifraKlijenta.*

9.3 Metode za prosljeđivanje parametara

Prilikom pozivanja potprograma mogu se odrediti stvarni parametri koji će se proslijediti koristeći pritom tri moguće metode prosljeđivanja parametara:

- Pozicijsko prosljeđivanje (eng. *Positional notation*)
- Imenovano prosljeđivanje (eng. *Named notation*)
- Mješovito prosljeđivanje (eng. *Mixed notation*)

Drugim riječima spomenute metode koristiti će se za uparivanje stvarnih i formalnih parametara.

Pozicijsko prosljeđivanje

U pozicijskom prosljeđivanju formalnom parametru dodjeljuje se vrijednost istim redoslijedom kako je deklarirano u potprogramu uzimajući u obzir da tip podataka također mora odgovarati. U slučaju da dođe do pogrešnog poretka parametara, dolazi do greške koju

je teško otkriti, osobito u slučaju ako se za parametre odaberu literali. Pozicijsko prosljeđivanje najčešća je metoda za prosljeđivanje parametara [38] [40] [41].

Primjer:

```
CREATE OR REPLACE FUNCTION pozicijsko_prosljedjivanje
(m NUMBER := 0, i NUMBER := 0, a NUMBER := 0 ) RETURN NUMBER IS
BEGIN
        RETURN m + i + a;
END;
/
```

Pozivanje procedure:

```
BEGIN
DBMS_OUTPUT.PUT_LINE('Zbroj: '|| pozicijsko_prosljedjivanje (7,6,3));
END;
/
```

Rezultat:

Zbroj: 16

PL/SQL procedure successfully completed.

Imenovano prosljeđivanje

Kod imenovanog prosljeđivanja formalni parametar (naziv parametra) eksplicitno se povezuje sa stvarnim parametrom (vrijednost parametra) koristeći pritom simbol =>. Također omogućuje da se u potprogramu proslijede vrijednosti samo obaveznim parametrima unatoč tomu što potprogram može sadržavati i neobavezne parametre. Sintaksa za imenovano prosljeđivanje je:

naziv_formalnog_parametra => vrijednost_argumenta

Kod imenovanog prosljeđivanja nije važan redoslijed parametara zato što se naziv formalnog parametra dodjeljuje eksplicitno, odnosno kod navođenja parametara nije potrebno paziti na isti redoslijed parametra u pozivnom programu i zaglavlju [38] [40] [41].

```
CREATE OR REPLACE FUNCTION imenovano_prosljedjivanje
(m NUMBER := 0, i NUMBER := 0, a NUMBER := 0 ) RETURN NUMBER IS
BEGIN
        RETURN m + i + a;
```

```
END;
/
```

Pozivanje procedure:

```
BEGIN
    DBMS_OUTPUT.PUT_LINE('Zbroj: ' || imenovano_prosljedjivanje (m => 6,i
=> 3,a => 7));
END;
/
```

Rezultat:

Zbroj: 16

PL/SQL procedure successfully completed.

Mješovito prosljeđivanje

Mješovito prosljeđivanje omogućuje pozivanje potprograma spajanjem pozicijskog i imenovanog prosljeđivanja. Kod ovog prosljeđivanja prvo treba koristiti pozicijsko prosljeđivanje, a tek onda imenovano zato što inače dolazi do greške u pozicijskom prosljeđivanju koju je teško otkriti pogotovo ako se za stvarne parametre koriste literali. Mješovito prosljeđivanje korisno je u slučajevima kada se u listi parametara na prvom mjestu nalaze definirani obavezni parametri, a zatim iza njih slijede neobavezni parametri [7] [14] [41].

```
CREATE OR REPLACE FUNCTION mjesovito_prosljedjivanje
(m NUMBER := 0, i NUMBER := 0, a NUMBER := 0 ) RETURN NUMBER IS
BEGIN
    RETURN m + i + a;
END;
/
```

Pozivanje procedure:

```
BEGIN
    DBMS_OUTPUT.PUT_LINE('Zbroj: ' || mjesovito_prosljedjivanje (7,a => 6,i
=> 3));
END;
/
```

Rezultat:

Zbroj: 16

PL/SQL procedure successfully completed.

9.4 Funkcije

Funkcija je potprogram sličan proceduri koji izračunava vrijednosti koje vraća pomoću RETURN klauzule koja se nalazi u izvršnom dijelu funkcije. Mogu prihvatiti jedan, više ili niti jedan parametar. Funkcije baš kao i procedure imaju jedinstveno ime pomoću kojega se pozivaju. Pohranjuju se u bazu podataka kao objekti baze podataka. Sintaksa za izradu funkcije je:

```
[CREATE [OR REPLACE]] FUNCTION naziv_funkcije
        [(naziv_parametra  [IN | OUT | IN OUT] podatkovni_tip [, ...])]
         RETURN podatkovni_tip
{IS | AS}
        [lokalne deklaracije]
BEGIN
        izvršne_naredbe
[EXCEPTION
        obrade_grešaka]
END [naziv_funkcije];
```

gdje *naziv_funckije* označava naziv same funkcije. [OR REPLACE] opcija omogućuje izmjenu postojeće funkcije. *naziv_parametara* sastoji se od izborne liste parametara koja sadrži ime i tip pridruženog parametra koji se definiraju kako bi se informacije proslijedile funkciji i poslale izvan funkcije nazad pozivnom programu. Nakon toga slijedi RETURN klauzula koja je kod funkcija obavezna i koja određuje tip podataka vrijednosti koju vraća funkcija. Tip podatka povratne vrijednosti deklarira se u zaglavlju funkcije. *lokalne deklaracije* omogućavaju deklariranje lokalnih identifikatora funkcije. Ako kod ne sadrži niti jednu deklaraciju, između ključnih riječi IS i BEGIN neće se nalaziti niti jedna naredba. i*zvršne_naredbe* su naredbe koje se izvršavaju kada ih funkcija pozove. Potrebno je imati barem jednu izvršnu naredbu nakon ključne riječi BEGIN i prije ključne riječi END ili EXCEPTION. *obrade_grešaka* navode dio za obradu grešaka [1] [7] [42]. Funkcija se unutar programa SQL Developer sprema u odjeljku *Functions.*

Primjer koda za model krojačnice:

```
CREATE OR REPLACE FUNCTION tkanina_info
```

```
        (sifra IN VARCHAR2) RETURN VARCHAR2
    AS
        ime VARCHAR2(80);
        var_sifra tkanina.sifra_tkanine%TYPE;
        tablica tkanina%ROWTYPE;
    BEGIN
        SELECT *
        INTO tablica
        FROM tkanina
        WHERE sifra_tkanine = sifra;

        ime := initcap(tablica.naziv_tkanine)||'  **  '||initcap(tablica.boja_tkanine);
        RETURN ime;

    EXCEPTION
        WHEN OTHERS THEN
    RAISE_APPLICATION_ERROR (-20001,'Dogodila se greška - '||SQLCODE||' - broj
    greške - '||SQLERRM);
    END;
    /
```

Pozivanje funkcije:

```
    SELECT tkanina_info(6) FROM dual;
```

Rezultat:

	TKANINA_INFO(6)
1	Platno ** Bijela

Slika 25 Prikaz informacija tkanine pod šifrom 6

Funkcija koja ne prima niti jedan parametar piše se bez zagrada. Funkcija može imati IN, OUT i IN OUT parametre ali preporučljivo je da se koriste samo IN parametri, pošto se korištenje preostala dva smatra lošom programerskom praksom.

Primjer koda za model krojačnice:

```
    CREATE OR REPLACE FUNCTION prosjeca_placa_krojac
     RETURN NUMBER AS
        prosjecna_placa NUMBER;
    BEGIN
        SELECT AVG(iznos_place)
```

```
        INTO prosjecna_placa
        FROM krojac;
        RETURN prosjecna_placa;
END prosjeca_placa_krojac;
/
```

Pozivanje funkcije:

```
SELECT prosjeca_placa_krojac FROM dual;
```

Rezultat:

	PROSJECA_PLACA_KROJAC
1	9846

Slika 26 Prikaz prosječne plaće za krojača - funkcija bez parametara

10. Paketi

Paket je shema objekta koja grupira PL/SQL tipove, konstante, varijable, potprograme, kursore i iznimke koji su logički povezani. To je jedna od prednosti paketa iz razloga što se spomenuti objekti mogu referencirati iz drugih PL/SQL blokova i time paketi omogućuju opciju korištenja globalnih varijabli. Pojednostavljeno, paketi su imenovani blokovi koji se pohranjuju kao objekt Oracle baze podataka koje program može referencirati ili ponovno koristiti kasnije. Paketi se unutar programa SQL Developer spremaju u odjeljku *Packages.* Ostale prednosti zbog kojih su paketi efikasni i korisni su [1] [7] [9]:

- Ponovno korištenje (eng. *Reusability*) – paket se kompajlira i pohranjuje u bazu podataka nakon što se kreira, odnosno kod koji se napiše može se ponovno koristiti od strane nekog drugog programa
- Preopterećenje (eng. *Overloading*) – unutar paketa mogu se kreirati jedna ili više procedura ili funkcija koje imaju isti naziv
- Modularnost (eng. *Modularity*) – Paketi omogućuju grupiranje objekata koji se koriste za stvaranje aplikacija. Svaki paket može se učiniti jednostavnim za razumijevanje isto kao i sučelja između paketa koja se mogu jasno i dobro definirati
- Bolje performanse (eng. *Better Performance*) – kada se prvi put pokrenu potprogrami paketa, Oracle baza podataka učitava cijeli paket u memoriju. Naknadno pozivanje drugih potprograma u istom paketu ne zahtjeva disk I/O. Paketi sprječavaju nepotrebno ponovno kompajliranje
- Sigurnost (eng. *Security*) – korištenje paketa dodaje dodatan sloj zaštite. U slučaju kada korisnik na primjer izvrši proceduru unutar paketa, procedura sadrži iste dozvole kao i vlasnik paketa. Paketi omogućuju izradu privatnih funkcija i procedura koje se mogu pozvati samo iz drugih funkcija i procedura unutar paketa što potiče skrivanje informacija

10.1 Arhitektura paketa

Paket je sastavljen od dviju temeljnih komponenti:

- Specifikacija paketa (eng. *Package Specification*) poznatija i kao zaglavlje paketa (eng. *Package header)*

- Tijelo paketa (eng. *Package body*) poznatije i kao definicija paketa (eng. *Package definition*)

10.1.1 Specifikacija paketa

Specifikacija paketa je sučelje (eng. *Interface*) paketa u kojem se deklariraju javni objekti[26] (varijable, kursori, procedure, funkcije i iznimke) koji su dostupni izvan paketa. Odnosno specifikacija paketa sadrži informacije o sadržaju paketa ali ne sadrži kod za potprograme. Za specifikaciju paketa vrijede ista pravila sintakse kao i za deklarativni dio[27] osim kod deklaracije funkcije ili potprograma. Neka od tih pravila su da se elementi paketa mogu pojaviti u bilo kojem redoslijedu s naglaskom da se objekt mora deklarirati prije nego se referencira. Nije nužno da svi tipovi elemenata budu prisutni. Sintaksa za specifikaciju paketa je:

```
CREATE [OR REPLACE] PACKAGE naziv_paketa AS -- specifikacija
        -- deklaracije javnih tipova i objekata
        -- specifikacije potprograma
END [naziv_paketa];
```

gdje se CREATE PACKAGE naredba koristi za definiranje novih specifikacija paketa. Pomoću ključne riječi REPLACE može se rekonstruirati postojeći paket. Unutar specifikacije paketa mogu se definirati novi tipovi, deklarirati globalne varijable, objekti, iznimke, kursori, procedure i funkcije.

Primjer koda za model krojačnice:

```
CREATE OR REPLACE PACKAGE krojac_placa AS
  FUNCTION dohvati_ime(sifraKrojaca NUMBER)
    RETURN VARCHAR2;

  FUNCTION dohvati_placu(sifraKrojaca NUMBER)
    RETURN NUMBER;
END krojac_placa;
```

[26] **Javnim objektima** nazivaju se svi objekti paketa koji se nalaze unutar specifikacije paketa, dok se **privatnim objektom** nazivaju potprogrami koji se ne nalaze u specifikaciji paketa ali se njihov kod nalazi u tijelu paketa i kojemu mogu pristupiti samo drugi potprogrami paketa.

[27] Specifikacija paketa je po strukturi ista kao i deklaracijski dio PL/SQL bloka ali treba naglasiti da specifikacija paketa ne smije sadržavati nikakav implementacijski kod.

U primjeru za specifikaciju paketa kreirao se paket naziva *krojac_placa* koji se sastoji od dvije funkcije *dohvati_ime* i *dohvati_placu* koje za parametre primaju šifru krojača.

10.1.2 Tijelo paketa

Izvršni kod za objekte opisane u paketu specifikacija sadrži tijelo paketa. Također može sadržavati i kod za objekte koji nisu deklarirani u specifikaciji, a ti se objekti nazivaju privatnim objektima i mogu se pozvati samo unutar paketa. Tijelo paketa zasebni je objekt od zaglavlja paketa te se ne može uspješno kompajlirati ako prethodno zaglavlje paketa nije bilo uspješno kompajlirano. Sintaksa za tijelo paketa je:

```
CREATE [OR REPLACE] PACKAGE BODY naziv_paketa AS -- tijelo
        -- deklaracije privatnih tipova i objekata
        -- tijela potprograma
[BEGIN
        -- naredbe za inicijalizaciju]
[EXCEPTION
        -- obrada iznimaka]
END [naziv_paketa];
```

Iz prikazane sintakse vidljivo je da je sintaksa dosta slična sintaksi za specifikaciju paketa osim što sadrži ključnu riječ BODY i implementirani kod specifikacije paketa. CREATE PACKAGE BODY naredba koristi se za definiranje tijela paketa. Pomoću ključne riječi REPLACE može se rekonstruirati postojeći paket. Tijelo paketa može sadržavati izborni dio za inicijalizaciju paketa koji se nalazi na kraju tijela paketa. Taj dio započinje ključnom riječju BEGIN, a završava ključnom riječju EXCEPTION ili END. Kada se aplikacija prvi puta referencira na objekt paketa, izvršava se odjeljak za inicijalizaciju paketa. Tijelo paketa nije obavezno, ali je dobra praksa koristiti ga. Izostavlja se u slučajevima kada zaglavlje paketa ne sadrži potprograme nego samo kursore, deklaraciju varijabli, tipove i tako dalje. Budući da su svi objekti u paketu vidljivi izvan paketa, ova tehnika je korisna za deklariranje globalnih varijabli i tipova. S druge strane tijelo paketa je obavezno u slučajevima kada:

- Specifikacija paketa sadrži deklaraciju kursora s RETURN klauzulom – u tijelo paketa potrebno je navesti SELECT naredbu
- Specifikacija paketa sadrži deklaraciju procedure ili funkcije – potrebno je dovršiti implementaciju spomenutog potprograma u tijelu paketa

- Kod se izvršava u dijelu za inicijalizaciju paketa – taj dio može se izvršiti samo u tijelu paketa zato što specijalizacija paketa ne podržava dio za inicijalizaciju paketa

Preostalo je još spomenuti par pravila koja su ključna u kodu tijela paketa:

- Kursori i potprogrami moraju imati isto zaglavlje i definiciju kako je navedeno i u specifikaciji paketa
- Deklaracija varijabli, iznimaka, konstanti i tipova ne bi se trebali ponavljati u tijelu paketa
- U tijelu paketa može se referencirati na bilo koji objekt koji je deklariran u specifikaciji. Za pristupanje bilo kojem tipu, objektu ili potprogramu paketa koristi se notacija točke, a sintaksa glasi *naziv_paketa.naziv_procedure.* Za brisanje paketa koristi se naredba DROP PACKAGE.

Primjer koda za model krojačnice – nastavak:

```
CREATE OR REPLACE PACKAGE BODY krojac_placa AS

        FUNCTION dohvati_ime(sifraKrojaca NUMBER) RETURN VARCHAR2 IS
                var_ime VARCHAR2(50);
BEGIN
                SELECT ime_krojaca || ' ' ||  prezime_krojaca
                INTO var_ime
                FROM krojac
                WHERE sifra_krojaca = sifraKrojaca;

                RETURN var_ime;

EXCEPTION
        WHEN NO_DATA_FOUND THEN
                RETURN NULL;
        WHEN TOO_MANY_ROWS THEN
                RETURN NULL;
END;

        FUNCTION dohvati_placu(sifraKrojaca NUMBER) RETURN NUMBER IS
                var_placa NUMBER(9,2);
BEGIN
                SELECT iznos_place
                INTO var_placa
                FROM krojac
```

```
            WHERE sifra_krojaca = sifraKrojaca;

            RETURN var_placa;

    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RETURN NULL;
        WHEN TOO_MANY_ROWS THEN
            RETURN NULL;
  END;
END krojac_placa;
```

Unutar tijela paketa nalazi se kod za implementaciju funkcija koje su prethodno deklarirane u specifikaciji paketa.

Pozivanje paketa vrši se pozivanjem njegovih funkcija:

```
DECLARE
    var_placa NUMBER(9,2);
    var_ime   VARCHAR2(50);
    sifraKrojaca NUMBER := &sifra_krojaca;
BEGIN
    var_ime   := krojac_placa.dohvati_ime(sifraKrojaca);
    var_placa := krojac_placa.dohvati_placu(sifraKrojaca);

    IF var_ime  IS NOT NULL AND
        var_placa IS NOT NULL
    THEN
        DBMS_OUTPUT.PUT_LINE ('Ime krojaca: ' || var_ime);
        DBMS_OUTPUT.PUT_LINE ('Iznos place:' || var_placa);
    END IF;
END;
```

Rezultat:

Ime krojaca: Snježana Gožović
Iznos place: 3750 kuna.

PL/SQL procedure successfully completed.

11. Sekvence

Sekvenca je objekt baze podataka koji automatski generira numerički niz prema pravilima koja su navedena u specifikaciji sekvence. Uzlazni i silazni redoslijed koristi se za kreiranje vrijednosti sekvence što bi značilo da je redoslijed bitan. Sekvenca se može koristiti za automatsko kreiranje vrijednosti primarnog ključa. Broj iz sekvence generira se neovisno o tablicama, što omogućuje da se isti broj može koristiti za jednu ili više tablica. Nakon kreiranja sekvence, njezinim vrijednostima u SQL naredbi može se pristupiti pomoću pseudo stupaca[28] [43] [44] [45] [46]:

- CURRVAL – vraća trenutnu vrijednost sekvence
- NEXTVAL – povećava slijed i vraća novu vrijednost

Parametri koje sekvenca može poprimiti su:

Naziv parametra	Opis
START WITH	Definira prvi broj u sekvenci (zadana vrijednost je 1)
MINVALUE	Definira najmanju vrijednost sekvence kada sekvenca ima negativan porast
NOMINVALUE	Minimalna vrijednost za ulazni niz je 1, a za silazni slijed je – 1026
MAXVALUE	Definira najveću vrijednost sekvence
NOMAXVALUE	Naznačuje maksimalnu vrijednost od 4611686018427387903 za uzlazni niz ili minimalnu vrijednost za silazni niz koja iznosi – 1
INCREMENT BY	Definira porast, odnosno za koliko se povećava sljedeći broj u sekvenci. Vrijednost cijelog broja može biti pozitivna (uzlazni niz) ili negativna (silazni niz) ali ne može biti 0. Zadana vrijednost je 1
CYCLE	Sekvenca se pokreće ispočetka kada dosegne maksimalnu ili minimalnu vrijednost

[28] U Oraclu pseudo stupac ponaša se kao stupac tablice ali on zapravo nije pohranjen u tablici. Vrijednosti pseudo stupca ne mogu se umetnuti, ažurirati ili brisati [54].

NOCYCLE	Sekvenca se neće pokrenuti ispočetka nakon što dosegne maksimalnu ili minimalnu vrijednost
CACHE	Određuje koliko će se brojeva čuvati u predmemoriji radi bržeg pristupa sekvenci. Broj mora biti cijeli bez predznaka. Zadana vrijednost je 20
NOCACHE	Koristi se kako bi se naznačilo da vrijednosti sekvence nisu bile unaprijed dodijeljene. Zadana vrijednost je 20
RESET BY	Određuje da tijekom ponovnog pokretanja baze podataka, baza podataka automatski izvršava podupit i vrijednost sekvence ponovno se pokreće s vraćenom vrijednošću
ORDER	Koristi se kako bi se osiguralo da se brojevi sekvence kreiraju redoslijedom kojim su traženi. Korisno je kada se brojevi sekvence koriste kao vremenske oznake
NOORDER	Koristi se kako bi se osiguralo da se brojevi sekvence ne kreiraju redoslijedom kojim su traženi. To je zadana postavka

Tablica 10 Prikaz parametara sekvence

Za kreiranje sekvence koristi se sintaksa:

```
CREATE SEQUENCE naziv_sekevnce
[INCREMENT BY porast]
[START WITH pocetna_vrijednost]
[MAXVALUE najveca_vrijednost | NOMAXVALUE]
[MINVALUE najmanja_vrijednost | NOMINVALUE]
[CYCLE | NOCYCLE]
[CACHE vrijednost | NOCACHE]
[ORDER | NOORDER];
```

Primjer koda za model krojačnice:

```
CREATE SEQUENCE povecaj_za_jedan
START WITH 1
INCREMENT BY 1;
```

```
INSERT INTO tkanina(sifra_tkanine, naziv_tkanine, boja_tkanine)
VALUES(povecaj_za_jedan.NEXTVAL, 'Svila', 'Bijela');
INSERT INTO tkanina(sifra_tkanine, naziv_tkanine, boja_tkanine)
VALUES(povecaj_za_jedan.NEXTVAL, 'Pliš', 'Siva');
```

Rezultat:

	SIFRA_TKANINE	NAZIV_TKANINE	BOJA_TKANINE
1	1	Svila	Bijela
2	2	Pliš	Siva

Slika 27 Prikaz unosa podataka u tablicu TKANINA

U primjeru sekvenca se je koristila za unos podataka u tablicu TKANINA, odnosno korištenjem parametra START WITH šifra tkanine se povećavala za jedan prilikom svakog novog unosa.

12. Okidači

Okidači (eng. *Triggers*) imenovani su PL/SQL blokovi pohranjeni kao samostalni objekti u bazi podataka (ne kao dio bloka ili paketa) koji se izvršavaju („*okidaju*“) implicitno kao odgovor na definirani događaj. U slučaju da se okidač onemogući, Oracle stroj (eng. *Oracle engine*) ih eksplicitno izvršava. Slični su procedurama ili funkcijama zato što također sadrže deklarativni dio, izvršni dio i dio bloka za obradu iznimaka, a razlikuju se po tome što se ne mogu pozvati direktno. Okidači ne primaju argumente i definiraju se nad tablicom, shemom, pogledom ili bazom podataka s kojim je događaj povezan. Događaj može biti bilo koji od sljedećih:

- DML (eng. *Database manipulation language*) naredbe – DELETE, INSERT, UPDATE; izvršavaju se kada se zapis umetne, ažurira ili obriše iz tablice. Mogu se koristiti i za provjeru valjanosti, za postavljanje zadanih vrijednosti, za onemogućavanje određenih DML operacija i tako dalje
- DDL (eng. *Database definition language*) naredbe – CREATE, ALTER, DROP; izvršavaju se kada neka DDL naredba koja se uspješno izvršila aktivira okidač (na primjer kod kreiranja tablice)
- Operacija nad bazom (eng. *Database operation*) - SERVERERROR, LOGON, LOGOFF, STARTUP, SHUTDOWN; izvršavaju se kada se baza podataka pokrene ili isključi ili kada se korisnik prijavi ili odjavi ili kada se pojavi greška

Sve to okidače čini vrlo moćnim alatom za upravljanje bazom podataka, odnosno pomoću njih radnje se mogu preusmjeriti ili ograničiti. Okidači se unutar programa SQL Developer spremaju u odjeljku *Triggers*. Neke od situacija u kojima se okidači mogu pokazati vrlo korisnima su [7] [9] [14] [47] [48]:

- Automatsko kreiranje vrijednosti primarnog ključa
- Bilježenje svih promjena koje su se napravile unutar tablice
- Uklanjanje mogućnosti pojave nevažećih operacija
- Osiguravaju autorizaciju i sigurnost baze podataka

12.1 Arhitektura okidača

Okidač se sastoji od dva dijela:

- Deklaracija okidača
- Tijelo okidača

Unutar deklaracije navodi se kako i kada se okidač poziva. Tijelo okidača ima istu strukturu kao i anonimni PL/SQL blok. Deklaracija okidača sastoji se od komponenti:

- Naziva okidača (eng. *Trigger name*) – mora biti jedinstveno, ali neki drugi objekt u shemi može imat isti naziv zato što okidači imaju vlastiti identifikator koji određuje jedinstvenu listu za naziv okidača
- Naredbe/događaji okidača (eng. *Trigger Statement/Event*) – naredbe koje uzrokuju pojavu okidača. To mogu biti DDL naredbe, DML naredbe ili bilo koja operacija nad bazom čije izvođenje dovodi do izvršavanja okidača
- Ograničenja okidača (eng. *Trigger Restriction*) – odnosi se na bilo koji uvjet ili ograničenje koje je nametnuto okidaču. Kao rezultat toga, ako je uvjet TRUE, pojavljuje se okidač, a ako je FALSE okidač se ne pojavljuje
- Liste naredbi (eng. *Trigger Action*) – to je ustvari tijelo okidača koje sadrži naredbe koje se trebaju izvršiti kada se okidač pojavi

Prve tri spomenute komponente odnose se na deklaraciju, a zadnja komponenta odnosi se na tijelo okidača.

12.2 DML okidači

U završnom radu obraditi će se samo DML okidači, dok se za preostale okidače detaljnije informacije mogu potražiti u Oracle dokumentaciji. DML okidač izvršava se pojavom bilo koje DML naredbe (DELETE, INSERT, UPDATE). DML okidač može se kreirati nad tablicom ili pogledom i oni su jedni od najčešće korištenih okidača. Mogu djelovati na sve retke ili samo na neke. Sintaksa za kreiranje okidača je:

```
CREATE [OR REPLACE] TRIGGER naziv_okidača
{BEFORE | AFTER} događaj_okidanja ON naziv_tablice
[FOR EACH ROW]
[WHEN (...)]
[DECLARE ... ]
        -- naredbe za deklariranje varijabli
 BEGIN
        -- izvršne naredbe
[EXCEPTION ... ]
```

```
        -- naredbe za obradu iznimaka
END [naziv_okidača];
```

gdje CREATE naredba označava kreiranje novog okidača. Naredbom REPLACE izmjenjuje se postojeći okidač i ta naredba je izborna, ali u većini slučajeva koristi se naredba CREATE zato što ukoliko se okidač pokušava izmijeniti bez ključne riječi REPLACE pojaviti će se poruka o grešci da u bazi podataka postoji neki drugi objekt s istim imenom. *naziv_okidača* označava naziv okidača. BEFORE i AFTER opcije označavaju kada će se okidač izvršiti, odnosno da li će to biti prije definiranog događaja odnosno naredbe ili poslije. Ne mogu se definirati nad pogledom. *događaj_okidanja* određuje DML naredbe koje će se primijeniti na okidač (INSERT, UPDATE, DELETE). Naredba UPDATE može se koristiti za cijeli zapis ili samo za listu stupaca odvojenu zarezima. *naziv_tablice* je naziv tablice koja je povezana s okidačem. Svaki DML okidač može se primijeniti samo na jednu tablicu. Klauzula FOR EACH ROW označava da li će se okidač aktivirati za svaki red koji je obuhvaćen DML naredbom. WHEN klauzula provjera da li je uvjet TRUE kako bi se okidač mogao izvršiti. WHEN klauzula nije obavezna te ako izostane, okidač će se svejedno pokrenuti. Sve spomenuto do sada naziva se zaglavljem okidača (od ključne riječi CREATE pa do WHEN klauzule). Od DECLARE naredbe počinje tijelo okidača koje je anonimni PL/SQL blok. Nije obavezan dio te sadrži kod za deklariranje varijabli. Ako se lokalne varijable ne trebaju deklarirati, naredba DECLARE se ne koristi. BEGIN klauzula označava izvršni dio okidača, odnosno mora sadržavati barem jednu naredbu te je obavezna. EXCEPTION klauzula označava dio za obradu iznimaka koji također nije obavezan. Na kraju koristi se END klauzula koja je obavezna naredba za okidač. Da bi se navelo koji okidač se zatvara, navodi se naziv okidača, odnosno *naziv_okidača* iza ključne riječi END.

Zajedno s DML okidačima mogu se koristiti i okidači na razini retka (eng. *Row-Level Triggers*). Oni omogućuju pristupanje novim i prethodnim vrijednostima iz svakog retka pomoću dva pseudo zapisa[29]:

- :NEW – sadrži nove vrijednosti
- :OLD – pohranjuje izvorne vrijednosti zapisa koje obrađuje okidač

Elementi pseudo zapisa su pseudo polja. Okidači imaju pristup vrijednostima stupca koje su ažurirane DML naredbama. Te vrijednosti stupca su elementi pseudo zapisa ili pseudo polja. Vrijednosti pseudo polja su oni stupci koji su umetnuti naredbom INSERT (sadrže samo nove

[29] Pseudo iz razloga što nemaju ista svojstva kao pravi PL/SQL zapisi.

vrijednosti pošto prethodne vrijednosti ne postoje), ažurirane naredbom UPDATE (sadrže nove i stare vrijednosti) i obrisani naredbom DELETE (sadrži samo prethodne vrijednosti pošto se nove ne upisuju).

Primjer koda za model krojačnice:

```
CREATE OR REPLACE TRIGGER min_placa
        BEFORE INSERT OR UPDATE OF iznos_place
        ON krojac
        FOR EACH ROW
BEGIN
        IF :new.iznos_place < 3750 THEN
                RAISE_APPLICATION_ERROR(-20100, 'Unesi barem minimalnu
plaću(3750 kuna)!!');
        END IF;
   END min_placa;
 /

INSERT INTO krojac VALUES (13,'Vlatka','Bakarić','Žensko','1993-06-09','Viša
stručna spremaa',3500.00,4);
```

Rezultat:

```
Error starting at line : 2.029 in command -
INSERT INTO KROJAC VALUES (13,'Vlatka','Bakarić','Žensko','1993-06-09','Viša
stručna spremaa',3500.00,4)
Error report -
ORA-20100: Unesi barem minimalnu plaću(3750 kuna)!!
ORA-06512: at "SYSTEM.MIN_PLACA", line 3
ORA-04088: error during execution of trigger 'SYSTEM.MIN_PLACA'
```

U primjeru za okidač postavljen je uvjet da se prije naredbe INSERT ili UPDATE provjeri varijabla *iznos_place* čija vrijednost ne smije biti manja od 3750 kako je zadano u uvjetu. Ako se unese manja vrijednost prikaže se pripadajuća poruka.

Primjer koda za model krojačnice:

```
CREATE OR REPLACE TRIGGER datum_rodjenja_krivo
        BEFORE INSERT OR UPDATE OF datum_rodjenja
        ON krojac
        FOR EACH ROW
BEGIN
        IF :new.datum_rodjenja >= TO_CHAR(SYSDATE, 'yyyy/mm/dd') THEN
```

```
                RAISE_APPLICATION_ERROR(-20100, 'Unesi ispravan datum
rođenja');
        END IF;
   END datum_rodjenja_krivo;
 /

  INSERT INTO KROJAC VALUES (13,'Vlatka','Bakarić','Žensko','2022-09-10','Viša
stručna spremaa',7000.00,4);
```

Rezultat:

Error starting at line : 2.048 in command -
INSERT INTO KROJAC VALUES (13,'Vlatka','Bakarić','Žensko','2022-09-10','Viša stručna spremaa',7000.00,4)
Error report -
ORA-20100: Unesi ispravan datum rođenja
ORA-06512: at "SYSTEM.DATUM_RODJENJA_KRIVO", line 3
ORA-04088: error during execution of trigger 'SYSTEM.DATUM_RODJENJA_KRIVO'

U primjeru za okidač postavljen je uvjet da se prije naredbe INSERT ili UPDATE provjeri varijabla *datum_rodjenja* za koju je postavljeno ograničenje da datum rođenja ne smije biti veći od današnjeg, u slučaju da je, okidač se izvrši te se prikaže pripadajuća poruka.

Okidač može biti u jednom od dva stanja, a ta stanje su ENABLED i DISABLED. Kada je okidač postavljen na ENABLED, on će se izvršiti ako su ispunjeni svi uvjeti, a kada je okidač postavljen na DISABLED, on se neće izvršiti unatoč tomu što su ispunjeni svi uvjeti. Prilikom kreiranja okidača ako se ne navede u kojem će biti stanju, prema zadanim postavkama to je uvijek stanje ENABLED. Promjene nad okidačem mogu se jedino izvršiti njegovom zamjenom ili ponovnim kreiranjem. Naredbom ALTER mijenja se stanje okidača, kompajliranje i preimenovanje:

ALTER TRIGGER *naziv_okidača* ENABLE – okidač će se izvršiti

ALTER TRIGGER *naziv_okidača* DISABLE – okidač se neće izvršiti

ALTER TRIGGER *naziv_okidača* COMPILE – kompajliranje okidača

ALTER TRIGGER *naziv_okidača* RENAME TO *novi_naziv_okidača* - preimenovanje okidača

Kod zamjene okidača koristi se CREATE TRIGGER naredba sa uključenom klauzulom OR REPLACE:

CREATE [OR REPLACE] TRIGGER *naziv_okidača*

Kod ponovnog kreiranja okidača, okidač se prvo mora obrisati naredbom DROP TRIGGER te ponovno kreirati naredbom CREATE TRIGGER.

13. Zaključak

Na kraju sa sigurnošću se može reći da PL/SQL nudi široku lepezu mogućnosti za manipulaciju podacima bilo to interno odnosno unutar samog Oraclea ili eksterno u vlastitoj aplikaciji. Kako je PL/SQL proceduralno proširenje SQL-a koji je najčešće korišteni jezik za upravljanje bazama podataka, PL/SQL ima mogućnost korištenja svih SQL manipulacija nad podacima i korištenje SQL funkcija. Sposobnošću PL/SQL-a da šalje blokove naredbi u bazu podataka, smanjuje se promet između aplikacije i same baze.

No koja je uopće budućnost PL/SQL-a? Kako je PL/SQL sastavni dio Oracle baze podataka, koja je sastavni dio poslovnih aplikacija širom svijeta[30], PL/SQL će još dugo ostati pri samom vrhu. Ali isto tako sve što se radi unutar PL/SQL-a moguće je napraviti u drugim programskim jezicima ili alatima. Isto tako postoje i drugi sustavi za upravljanje bazama podataka poput MS SQL Servera koji je lakši i jednostavniji za korištenje i koji se može koristiti za izgradnju, implementaciju i upravljanje aplikacijama koje se nalaze lokalno ili u oblaku. Neovisno koji se sustav za upravljanje relacijskim bazama koristi, temeljno poznavanje PL/SQL i SQL-a jedno je od traženijih vještina koje developeri u današnje vrijeme trebaju posjedovati.

[30] Prema posljednjim podacima 65 597 kompanija diljem svijeta koristi PL/SQL i većina tih kompanija nalazi se u Sjedinjenim Američkim Državama.

14. Literatura

[1] B. Rosenzweig i E. S. Rakhimov, Oracle PLSQL by Example (4th Edition), 2008.

[2] M. V. Academy, »Multisoft Virtual Academy,« 12 Siječanj 2016. [Mrežno]. Dostupno: https://www.multisoftvirtualacademy.com/blog/oracle-plsql-advantages-and-comparison-over-other-languages/. [Pokušaj pristupa 30 Listopad 2021].

[3] Tutorialspoint, »Tutorialspoint.com - PL/SQL - Overview,« [Mrežno]. Dostupno: https://www.tutorialspoint.com/plsql/plsql_overview.htm. [Pokušaj pristupa 30 Listopad 2021].

[4] M. Pavlić, Oblikovanje baza podataka, Rijeka: Odjel za informatiku Sveučilišta u Rijeci, 2011.

[5] M. Puriš, »Poslovna aplikacija za krojačnicu nad relacijskom bazom podataka - Clarion,« 2019.

[6] D. Jakšić i P. Poščić, »Clarion 10 - Uređivač Rječnika,« Rijeka.

[7] S. Feuerstein i B. Pribyl, Oracle PL/SQL Programming, Sixth Edition, 2014.

[8] O. Tutorial, »Oracle Tutorial - PL/SQL Anonymous Block,« [Mrežno]. Dostupno: https://www.oracletutorial.com/plsql-tutorial/plsql-anonymous-block/. [Pokušaj pristupa 4 Studeni 2021].

[9] S. Urman, R. Hardman i M. McLaughlin, Oracle Database 10g PL/SQL Programming 1st Edition, 2004.

[10] M. Rosenblum i P. Dorsey, Oracle PL / SQL For Dummies 1st Edition, 2006.

[11] Oracle, »Oracle.com - PL/SQL Data Types,« [Mrežno]. Dostupno: https://docs.oracle.com/cd/B28359_01/appdev.111/b28370/datatypes.htm#i10924 . [Pokušaj pristupa 13 Studeni 2021].

[12] Oracle, »Oracle.com - Predefined PL/SQL Numeric Data Types and Subtypes,« 2009. [Mrežno]. Dostupno: https://docs.oracle.com/cd/B28359_01/appdev.111/b28370/datatypes.htm#i46029. [Pokušaj pristupa 13 Studeni 2021].

[13] Oracle, »Oracle.com - Predefined PL/SQL BOOLEAN Data Type,« 15 Studeni 2009. [Mrežno]. Dostupno: https://docs.oracle.com/cd/B28359_01/appdev.111/b28370/datatypes.htm#i45907. [Pokušaj pristupa 2021].

[14] M. McLaughlin, Oracle Database 12c PL/SQL Programming, 2014.

[15] Oracle, »Oracle - Types of LOBs,« [Mrežno]. Dostupno: https://docs.oracle.com/en/database/oracle/oracle-database/21/adlob/LOB-classifications.html#GUID-0A692C1B-1C95-4121-8F95-25BE465B87F6. [Pokušaj pristupa Studeni 20211].

[16] Oracle, »Oracle.com - LOB Classifications,« [Mrežno]. Dostupno: https://docs.oracle.com/en/database/oracle/oracle-database/21/adlob/LOB-classifications.html. [Pokušaj pristupa 16 Studeni 2021].

[17] Tutorialspoint, »Tutorialspoint.com - PL/SQL - Variables,« [Mrežno]. Dostupno: https://www.tutorialspoint.com/plsql/plsql_variable_types.htm. [Pokušaj pristupa 17 Studeni 2021].

[18] Y. Peng, »demo2s.com - Oracle PL/SQL %ROWTYPE,« [Mrežno]. Dostupno: https://www.demo2s.com/oracle/oracle-pl-sql-rowtype.html. [Pokušaj pristupa 18 Studeni 2021].

[19] Oracle, »Oracle.com - PL/SQL Language Fundamentals,« [Mrežno]. Dostupno: https://docs.oracle.com/cd/E11882_01/appdev.112/e25519/fundamentals.htm#LNPLS002. [Pokušaj pristupa 20 Studeni 2021].

[20] Oracle, »Oracle.com - Operators, Functions, Expressions, Conditions,« [Mrežno]. Dostupno: https://docs.oracle.com/cd/A58617_01/server.804/a58225/ch3all.htm. [Pokušaj pristupa 20 Studeni 2021].

[21] S. Technologies, »Studytonight - Operators in PL/SQL,« [Mrežno]. Dostupno: https://www.studytonight.com/plsql/plsql-operators. [Pokušaj pristupa 20 Studeni 2021].

[22] Tutorialspoint, »Tutorialspoint.com - PL/SQL - Operators,« [Mrežno]. Dostupno: https://www.tutorialspoint.com/plsql/plsql_operators.htm. [Pokušaj pristupa 21 Studeni 2021].

[23] S. Technologies, »Studytonight - PL/SQL Conditional Statements,« [Mrežno]. Dostupno: https://www.studytonight.com/plsql/decision-making-plsql. [Pokušaj pristupa 24 Studeni 2021].

[24] Y. Peng, »demo2s.com - Oracle PL/SQL Conditional Evaluation,« [Mrežno]. Dostupno: https://www.demo2s.com/oracle/oracle-pl-sql-conditional-evaluation-via-if-then.html. [Pokušaj pristupa 25 Studeni 2021].

[25] Oracle, »Oracle.com - Using PL/SQL Control Structures,« [Mrežno]. Dostupno: https://docs.oracle.com/cd/B28359_01/appdev.111/b28370/controlstructures.htm#LNPLS004. [Pokušaj pristupa 25 Studeni 2021].

[26] Y. Peng, »demo2s.com - Oracle PL/SQL Sequential Navigation using GOTO,« [Mrežno]. Dostupno: https://www.demo2s.com/oracle/oracle-pl-sql-sequential-navigation-using-goto.html. [Pokušaj pristupa 3 Prosinac 2021].

[27] O. Tutorial, »Oracle Tutorial.com - PL/SQL GOTO Statement,« [Mrežno]. Dostupno: https://www.oracletutorial.com/plsql-tutorial/plsql-goto/. [Pokušaj pristupa 3 Prosinac 2021].

[28] D. Jakšić, »Merlin,« [Mrežno]. Dostupno: https://moodle.srce.hr/2019-2020/pluginfile.php/3385308/mod_resource/content/0/2.%20PL_SQL_Kursori_Iznimke_Procedure.pdf. [Pokušaj pristupa 7 Prosinac 2021].

[29] S. Technologies, »Studytonight - Cursor in PL/SQL,« [Mrežno]. Dostupno: https://www.studytonight.com/plsql/plsql-cursor. [Pokušaj pristupa 7 Prosinac 2021].

[30] TutorialsPoint, »TutorialsPoint - PL/SQL - Cursors,« [Mrežno]. Dostupno: https://www.tutorialspoint.com/plsql/plsql_cursors.htm. [Pokušaj pristupa 7 Prosinac 2021].

[31] O. Tutorial, »Oracle Tutorial - PL/SQL Cursor,« [Mrežno]. Dostupno: https://www.oracletutorial.com/plsql-tutorial/plsql-cursor/. [Pokušaj pristupa 9 Prosinac 2021].

[32] O. Tutorial, »Oracle Tutorial - PL/SQL Cursor with Parameters,« [Mrežno]. Dostupno: https://www.oracletutorial.com/plsql-tutorial/plsql-cursor-with-parameters/. [Pokušaj pristupa 9 Prosinac 2021].

[33] O. Tutorial, »Oracle Tutorial - PL/SQL Record,« [Mrežno]. Dostupno: https://www.oracletutorial.com/plsql-tutorial/plsql-record/. [Pokušaj pristupa 28 Siječanj 2022].

[34] Oracle.com, »Database PL/SQL Language Reference - PL/SQL Collections and Records,« [Mrežno]. Dostupno: https://docs.oracle.com/en/database/oracle/oracle-database/19/lnpls/plsql-collections-and-records.html#GUID-7115C8B6-62F9-496D-BEC3-F7441DFE148A. [Pokušaj pristupa 25 Siječanj 2022].

[35] O. Tutorial, »Oracle Tutorial.com - PL/SQL Collections,« [Mrežno]. Dostupno: https://www.oracletutorial.com/plsql-tutorial/plsql-associative-array/. [Pokušaj pristupa 25 Siječanj 2022].

[36] Oracletutorial, »Oracletutorial - PL/SQL Exception Propagation,« [Mrežno]. Dostupno: https://www.oracletutorial.com/plsql-tutorial/plsql-exception-propagation/. [Pokušaj pristupa 20 Travanj 2022].

[37] O. H. Center, »Oracle - PL/SQL Subprograms,« [Mrežno]. Dostupno: https://docs.oracle.com/en/database/oracle/oracle-database/19/lnpls/plsql-subprograms.html#GUID-13BEBBEC-02D4-48E8-A059-DFEAC4751A3B. [Pokušaj pristupa 21 Travanj 2022].

[38] Tutorialspoint, »Tutorialspoint - PL/SQL - Procedures,« [Mrežno]. Dostupno: https://www.tutorialspoint.com/plsql/plsql_procedures.htm. [Pokušaj pristupa 21 Travanj 2022].

[39] O. H. Center, »Oracle - Subprogram Parameter Modes,« [Mrežno]. Dostupno: https://docs.oracle.com/en/database/oracle/oracle-database/19/lnpls/plsql-subprograms.html#GUID-518B8827-26CC-4734-B799-ACB038185638. [Pokušaj pristupa 2023 Travanj 2022].

[40] O. H. Center, »Oracle - Positional, Named, and Mixed Notation for Actual Parameters,« [Mrežno]. Dostupno: https://docs.oracle.com/en/database/oracle/oracle-database/19/lnpls/plsql-subprograms.html#GUID-A5DA8CF5-1BCC-4ABE-9B68-DB593FF1D2CC. [Pokušaj pristupa 24 Travanj 2022].

[41] M. Sharma, »Rebellion Rider - Calling Notation For PL/SQL Subroutines In Oracle Database,« [Mrežno]. Dostupno: http://www.rebellionrider.com/calling-notation-for-pl-sql-subroutines-in-oracle-database/. [Pokušaj pristupa 24 Travanj 2022].

[42] Tutorialspoint, »Tutorialspoint - PL/SQL - Functions,« [Mrežno]. Available: https://www.tutorialspoint.com/plsql/plsql_functions.htm. [Pokušaj pristupa 25 Travanj 2022].

[43] Oracle, »Oracle - Create Sequence,« [Mrežno]. Dostupno: https://docs.oracle.com/cd/B12037_01/server.101/b10759/statements_6014.htm. [Pokušaj pristupa 27 Travanj 2022].

[44] Oracletutorial.com, »Oracletutorial - Oracle Sequence,« [Mrežno]. Dostupno: https://www.oracletutorial.com/oracle-sequence/. [Pokušaj pristupa 27 Travanj 2022].

[45] R. Bennett, »Guru99 - SAP HANA Tutorial: Create Sequence,« [Mrežno]. Dostupno: https://www.guru99.com/sap-hana-create-sequence.html. [Pokušaj pristupa 27 Travanj 2022].

[46] Oracletutorial.com, »Oracletutorial - Oracle Create Sequence,« [Mrežno]. Dostupno: https://www.oracletutorial.com/oracle-sequence/oracle-create-sequence/. [Pokušaj pristupa 27 Travanj 2022].

[47] R. Peterson, »Guru99 - Oracle PL/SQL Trigger Tutorial: Instead of, Compound [Example],« [Mrežno]. Dostupno: https://www.guru99.com/triggers-pl-sql.html. [Pokušaj pristupa 4 Svibanj 2022].

[48] O. H. Center, »Oracle - Trigger Design Guidelines,« [Mrežno]. Dostupno: https://docs.oracle.com/en/database/oracle/oracle-database/19/lnpls/plsql-triggers.html#GUID-78B64011-B288-4EDD-B595-2DDBB3835ACA. [Pokušaj pristupa 4 Svibanj 2022].

[49] Oracle, »Oracle.com - DBMS_SQL,« [Mrežno]. Dostupno: https://docs.oracle.com/database/121/TTPLP/d_sql.htm#TTPLP058. [Pokušaj pristupa 5 Studeni 2021].

[50] Oracle, »Oracle.com - What is a Relational Database,« [Mrežno]. Dostupno: https://www.oracle.com/in/database/what-is-a-relational-database/. [Pokušaj pristupa 27 Listopad 2021].

[51] Oracle, »Oracle.com - Using PL/Scope,« [Mrežno]. Dostupno: https://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_plscope.htm#ADFNS022. [Pokušaj pristupa 5 Studeni 2021].

[52] IBM, »IBM Db2 Big SQL,« [Mrežno]. Dostupno: https://www.ibm.com/docs/en/db2-big-sql/7.0?topic=list-national-character-strings. [Pokušaj pristupa 14 Studeni 2021].

[53] Oracle, »Oracle.com - Supporting Multilingual Databases with Unicode,« [Mrežno]. Dostupno: https://docs.oracle.com/cd/B19306_01/server.102/b14225/ch6unicode.htm. [Pokušaj pristupa 16 Studeni 2021].

[54] GeeksforGeeks, »GeeksforGeeks - Pseudocolumn in Oracle SQL,« [Mrežno]. Dostupno: https://www.geeksforgeeks.org/pseudocolumn-oracle-sql/. [Pokušaj pristupa 27 Travanj 2022].

15. Popis slika

16. Popis tablica