Računarstvo i informatika

Katedra za računarstvo
Elektronski fakultet u Nišu

Sistemi baza podataka
Napredni SQL (II deo)

Letnji semestar 2015

Sadržaj

- Procedure i funkcije
- Paketi
- Trigeri
- Dinamički SQL
- Sekvence
- Slogovi
- Kolekcije
- Objekti

Procedure i funkcije

- PL/SQL raspolaže mehanizmima koji omogućavaju deklarisanje funkcija i procedura kao samostalnih PL/SQL blokova.
- Funkcije i procedure se mogu koristiti u drugim PL/SQL blokovima (anonimnim blokovima ili drugim funkcijama i procedurama).
- Sintaksa za kreiranje procedure:

CREATE [OR REPLACE] PROCEDURE <ime procedure> [(<lista parametara>)]
IS
<deklaracije promenljivih>
BEGIN
< sekvenca PL/SQL naredbi >
[**EXCEPTION**
< sekvenca PL/SQL naredbi za obradu grešaka>]
END [<ime procedure>]**;**

Procedure i funkcije

- Sintaksa za kreiranje funkcije:

CREATE [OR REPLACE] FUNCTION <ime funkcije> [(<lista parametara>)]
RETURN <tip podatka koji funkcija vraća> **IS**
<deklaracije promenljivih>
BEGIN
< sekvenca PL/SQL naredbi >
[**EXCEPTION**
< sekvenca PL/SQL naredbi za obradu grešaka>]
END [<ime funkcije>]**;**

- Opciono se koristi klauzula **OR REPLACE** kojom se modifikuje već postojeća procedura ili funkcija.
- Klauzula **DECLARE** se za razliku od anonimnih PL/SQL blokova ne može koristiti unutar tela procedura ili funkcija.

Procedure i funkcije

- Za brisanje procedura i funkcija koriste se odgovarajuće **DROP** naredbe.

DROP PROCEDURE <ime procedure>;
DROP FUNCTION <ime funkcije>;

- Za parametre procedura i funkcija mogu se koristiti svi validni PL/SQL ili ORACLE tipovi podataka.
- Umesto eksplicitnih tipova podataka mogu se koristiti i implicitne deklaracije u formi %TYPE ili %ROWTYPE.
- Za tipove podataka NUMBER, CHAR ili VARCHAR ne smeju se specificirati podaci o dužini i preciznosti. (**Npr. Ne sme se parametar definisati kao NUMBER(6, 3) već samo kao NUMBER**).

Procedure i funkcije

- Sintaksa za deklarisanje parametara:

<ime parametra> [**IN** | **OUT** | **IN OUT**] <tip> [{**:=** | **DEFAULT**} <PL/SQL izraz>];

- Opcione klauzule **IN**, **OUT** i **IN OUT** definišu način na koji se parametar koristi:
  - IN – podrazumevani mod parametra. Parametar se može koristiti unutar procedure ili funkcije ali mu se vrednost ne može menjati.
  - OUT – vrednost parametra se ne može koristiti unutar procedure ili funkcije ali mu se može dodeliti vrednost.
  - IN OUT – kombinacija prethodna dva režima. Vrednost parametra se može koristiti ali mu se unutar funkcije ili procedure može dodeliti i nova vrednost.

Procedure i funkcije

```
CREATE PROCEDURE povecaj_platu(odeljenje NUMBER,
                               procenat NUMBER DEFAULT 0.5)
IS
CURSOR radnici(brojod NUMBER)
SELECT PLATA FROM RADNIK WHERE BROD = brojod
FOR UPDATE OF PLATA;
plata NUMBER(8);
BEGIN
        OPEN radnici(odeljenje);
        LOOP
                FETCH radnici INTO plata;
                EXIT WHEN radnici%NOTFOUND;
                UPDATE RADNIK
                SET PLATA = PLATA * (1.0 + procenat)
                WHERE CURRENT OF radnici;
        END LOOP;
        CLOSE radnici;
        COMMIT;
END povecaj_platu;
```

Procedura menja platu svim radnicima koji rade u određenom odeljenju.

Broj odeljenja i procenat promene su ulazni parametri procedure.

Prilikom otvaranja kursora koristi se prosleđeni broj odeljenja.

Procedura se poziva kao

EXECUTE povecaj_platu(10, 3) ;

Ukoliko se poziva iz nekog PL/SQL programa ili druge funkcije EXSECUTE može da se izostavi.

Procedure i funkcije

```
CREATE FUNCTION suma_plata(odeljenje NUMBER)
RETURN NUMBER
IS
plata NUMBER(8);
BEGIN
    plata := 0.0;
    FOR radnik IN (SELECT PLATA
                   FROM RADNIK
                   WHERE BROD = odeljenje; )
    LOOP
        plata := plata + radnik.PLATA;
    END LOOP;
    RETURN plata;
END suma_plata;
```

Za funkciju se eksplicitno mora deklarisati tip koji vraća.

Da bi se funkcija izvršila potrebno je najpre deklarisati promenljivu koja će prihvatiti vrednost funkcije.

```
VARIABLE plata NUMBER;
EXECUTE :plata := suma_plata(10);
```

Paketi

- Kao i standardni programski jezici PL/SQL obezbeđuje mehanizme koji omogućavaju modularno programiranje.
- PL/SQL paketi omogućavaju grupisanje PL/SQL blokova, procedura i funkcija.
- PL/SQL paket se satoji od:
  - Specifikacije paketa (package specification) – definiše interfejse koji su vidljivi korisnicima i pomoću kojih se pozivaju i koriste funkcionalnosti paketa.
  - Telo paketa (package body) – implementira specificirane interfejse.
- Organizacija se jako slična C/C++ organizaciji (h i c/cpp datoteke).

Paketi

```
CREATE PACKAGE preduzece
AS
FUNCTION suma_plata(odeljenje NUMBER)  RETURN NUMBER;
PROCEDURE povecaj_platu(odeljenje NUMBER, procenat NUMBER DEFAULT 0.5)
...
END preduzece;

CREATE PACKAGE BODY preduzece
AS
FUNCTION suma_plata(odeljenje NUMBER)  RETURN NUMBER
IS
BEGIN
/*Sekvenca PL/SQL naredbi koja predstavlja telo funkcije*/
END suma_plata;
PROCEDURE povecaj_platu(odeljenje NUMBER, procenat NUMBER DEFAULT 0.5)
IS
BEGIN
/*Sekvenca PL/SQL naredbi koja predstavlja telo procedure
END povecaj_platu;
END preduzece;
```

Paketi

- Procedure i funkcije definisane u paketu se pozivaju korišćenjem sledeće sintakse:

[EXECUTE]<ime paketa>. {<ime procedure> | <ime funkcije>} [(<lista parametara>)]**;**

- Ukoliko se procedura ili funkcija poziva u nekom drugom PL/SQL bloku onda ključna reč EXECUTE može da se izostavi.
- Oracle nudi nekeliko predefinsanih paketa:
  - DBMS_OUTPUT
  - UTL_FILE
  - DBMS_JOB
  - DBMS_SQL
  - ...

Trigeri

- Element šeme baze podataka
- Blok PL/SQL koda koji se izvršava kao odgovor na događaje – insert, update, delete, ako je ispunjen zadati uslov
- Izvršenje – jednom, pre ili posle insert, delete or update.
- Nadgledanje (potencijalnih) promena u bazi podataka
  - Nakon promene, ili pre nje, izvršava se definisani kod
- Forsiranje ograničenja
  - Početna stanja i ograničenja tranzicije, ali i za pristup postojećim ograničenjima

Trigeri

- Primena:
  - Forsiranje poslovnih (business) pravila koja se ne mogu predstaviti built-in ograničenjima integriteta
  - **Automatsko setovanje IDova (autonumber, surogat ključeva) i izvedenih vrednosti (izvedeni atributi)**
  - Provera promena u podacima
  - Automatsko čuvanje starih verzija podataka (logovanje akcija korisnika)
  - Automatska propagacija modifikacija
  - Realizacija materijalizovanih pogleda (pogledi koji fizički postoje, fizička kopija podataka)i replikacija promena u osnovne tabele
  - Podrška za smart modifikacije pogleda

Trigeri

- Šta se može nadgledati:
  - **Događaji u bazi podataka**
    - Startup, Shutdown, Logon, Logoff, Server Errors
  - **Događaji koji se odnose na definiciju podataka**
    - Create, Alter, Drop, itd.
  - **Događaji koji se odnose na manipulaciju podacima**
    - Insert, Update, Delete
- Granularnost trigera:
  - **Na nivou vrste (Row-level granularity)**: promena na nivou jedne vrste inicirana događajem (jedan UPDATE izraz koji može da prouzrokuje više događaja)
  - **Na nivou iskaza (Statement-level granularity)**: događaju su iskazi (jedan UPDATE koji menja više vrsti u jednom događaju)

Trigeri

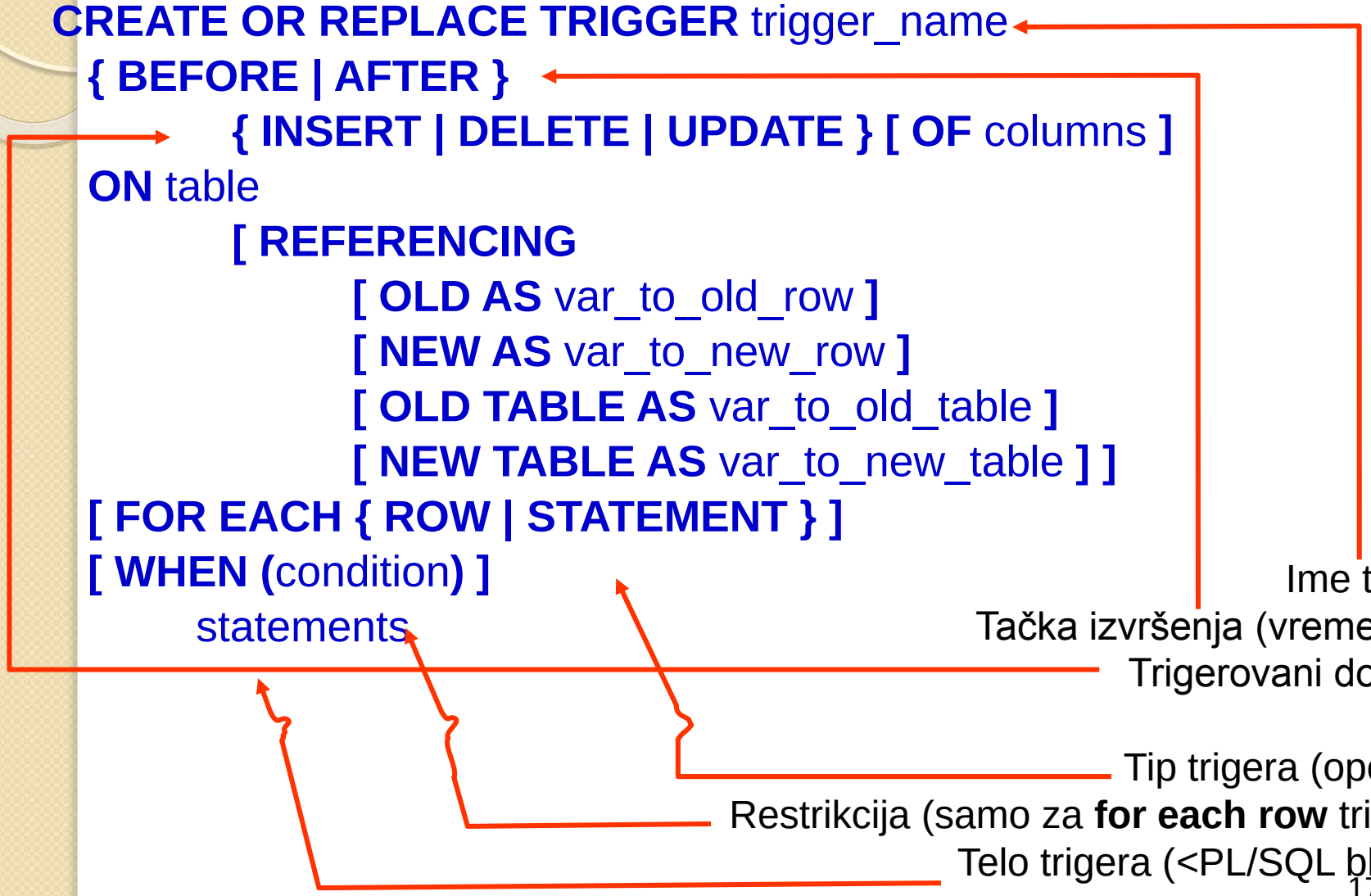
- Granularnost trigera:
  - Na nivou izraza / Statement Triger
    - Izvršava se kod trigera jednom, pre ili posle izraza koji je naveden
  - Na nivou vrste / Row Triger
    - Izvršava se kod trigera jednom, pre ili posle promene svake vrste (insert, update, ili delete)
    - Pristupa vrednostima atributa pre i posle promene u vrsti
- Višestruki trigeri – važan je izbor načina izvršenja!
  - Evaluacija jednog uslova u vremenskom trenutku i ako je ispunjen, odmah izvršiti akciju, ili
  - Evaluacija svih uslova, nakon toga izvršiti akcije. Izvršenje jedne akcije može da utiče na sledeće

Trigeri

- **Događaji**: INSERT, DELETE, ili UPDATE ili promene u individualnim vrstama prouzrokovane tim naredbama
- Uslov: bilo šta što je dozvoljeno za WHERE klauzulu
- **Akcija**: individualni SQL izraz ili program pisan korišćenjem PL/SQL-a
- **Razmatranje**: trenutno
  - Uslov se može odnositi na stanje posmatrane vrste ili tabele pre ili posle odigravanja događaja
- **Izvršenje**: trenutno
  - Može biti pre ili posle izvršenje trigerovanog događaja
  - Akcija za pre- triger ne može da modifikuje bazu podataka
- **Granularnost**: na nivou vrste ili iskaza

Trigeri

```
CREATE OR REPLACE TRIGGER trigger_name
 { BEFORE | AFTER }
        { INSERT | DELETE | UPDATE } [ OF columns ]
 ON table
          [ REFERENCING
                  [ OLD AS var_to_old_row ]
                  [ NEW AS var_to_new_row ]
                  [ OLD TABLE AS var_to_old_table ]
                  [ NEW TABLE AS var_to_new_table ] ]
 [ FOR EACH { ROW | STATEMENT } ]
 [ WHEN (condition) ]
        statements
```



Trigeri

- Definicija trigera se ne može promeniti, on se može samo ponovo kreirati korišćenjem REPLACE klauzule
- Brisanje trigera

  DROP TRIGGER <TRIGGER_NAME>
- Nakon kreiranja trigera, on je aktivan
- Aktivacija/dekativacija trigera:

  ALTER TRIGGER <TRIGGER_NAME> DISABLE | ENABLE

Trigeri

- **BEFORE trigeri se izvršavaju PRE DML operacije (INSERT, UPDATE, DELETE, MERGE)**
- **Trigerovani kod vidi stanje BP PRE/BEFORE modifikacije.**

- **AFTER trigeri se izvršavaju POSLE modifikacije**
- **Trigerovani kod vidi stanje BP POSLE/AFTER izvršenja operacije.**
- **Ako triger generiše izuzetak, modifikacija nije završena**

Trigeri

- Redosled izvršavanja trigera kod izvršenja DELETE/INSERT/UPDATE naredbi
  - Svi BEFORE statement trigeri koji mogu da se aktiviraju se izvršavaju u nedefinisanom redosledu
  - Izvršava se navedeni izraz (koji može da modifikuje jednu ili više vrsti)
  - Modifikacija svake vrste
    - Svi BEFORE Row trigeri se izvršavaju u nedefinisanom redosledu
    - Svi AFTER Row trigeri se izvršavaju u nedefinisanom redosledu
  - Svi AFTER Statement trigeri koji mogu da se aktiviraju se izvršavaju u nedefinisanom redosledu

Ako triger može da se izvrši i kao BEFORE ili AFTER triger, izborom tipa trigera možete da utičete na redolsed izvršenja, tako da se on izvrši pre ili posle drugih trigera

Trigeri

BEFORE ROW trigeri se izvršavaju

- **Posle izračunavanja nove vrednosti vrste**
- **Pre nego što se vrsta zapamti u tabeli**

Ako hoćete da modifikujete vrednost u vrsti koja se insert-uje ili update-uje, *morate* da koristite BEFORE triger

```
CREATE OR REPLACE TRIGGER limit_salary
   BEFORE UPDATE OF PLATA OR INSERT ON RADNIK
   FOR EACH ROW
BEGIN
   IF :NEW.PLATA  > 5000 THEN
      :NEW.PLATA  := 5000;
   END IF;
END;
```

Ako insert-ovani/update-ovani radnik ima PLATA > 5000, tada se njegova plata vraća na 5000

Trigeri

WHEN se može koristiti kod (BEFORE ili AFTER) trigera za vrstu da se ograniče stanja u kojima se triger izvršava, pri tome se povećavaju performanse

```
CREATE OR REPLACE TRIGGER limit_salary
    BEFORE INSERT OR UPDATE OF PLATA ON RADNIK
    FOR EACH ROW
    WHEN (new.PLATA > 5000)
BEGIN
   :NEW.PLATA  := 5000;
END;
```

Trigeri

Zabrana prekomernih promena plate

```
CREATE OR REPLACE TRIGGER emp_check_salary
   BEFORE UPDATE OF PLATA ON RADNIK
   FOR EACH ROW
   WHEN (new.PLATA > 1.1 * old.PLATA)
BEGIN
  :NEW.PLATA  := 1.1 * :OLD.PLATA;
END;
```

Trigeri

Zabrana prekomernih promena plate

```
CREATE OR REPLACE TRIGGER emp_check_salary
    AFTER UPDATE OF PLATA ON RADNIK
    FOR EACH ROW
CALL CheckSalary( :OLD.PLATA, :NEW.PLATA )
```

```
PROCEDURE CheckSalary(
      oldsal number, newsal number) IS
BEGIN
      IF newsal > 1.1*oldsal THEN
         RAISE_APPLICATION_ERROR( -20032,
            Povecanje plate nije dozvoljeno' );
      END IF;
END;
```

Napomena: u ovom slučaju se vrši rollback za ceo update izraz!!

Dinamički SQL

- Većina programa izvršava naredbe koje se tokom vremena ne menjaju.
- Postoje situacije u kojima nije unapred poznata čitava naredba ili neki deo naredbe koju je potrebno izvršiti.
- U takvim situacijam se naredba koju treba izvršiti dinamički formira u toku samog izvršavanja programa.
- Dinamički SQL omogućava izvršavanje SQL naredbi koje se menjaju prilikom svakog sledećeg pokretanja programa.

Dinamički SQL

- Kod PL/SQL-a dinamički SQL se koristi u sledećim situacijama:
  - kada je potrebno izvršiti DDL naredbe (PL/SQL ne dozvoljava direktno izvršavanje DDL naredbi u telu programa)
  - kada je potrebno povećati fleksibilnost programa (tokom izvršenja upita menjaju se nazivi tabela koje se referenciraju, menjaju se lista kolona koju upit referencira, ...)
- Za korišćenje dinamičkog SQL-a u PL/SQL programima je na raspolaganju EXECUTE IMMEDIATE klauzula.

Dinamički SQL

- EXECUTE IMMEDIATE ima sledeću sintaksu:

EXECUTE IMMEDIATE <dinamički kreirani string>
[**INTO** {<deklarisana promenljiva>[, <deklarisana promenljiva>]... | <slog>}]
[**USING** [**IN** | **OUT** | **IN OUT**] <parametar>
[, [**IN** | **OUT** | **IN OUT**] <parametar>]...];

- Dinamički kreirani string predstavlja validno definisanu SQL naredbu ili PL/SQL blok.
- Deklarisana promenljiva je PL/SQL promenljiva u koju će se smestiti vrednost koju SELECT naredba pribavlja iz baze podataka.

Dinamički SQL

```
DECLARE
  sql_izraz    VARCHAR2(100);
  plsql_blok VARCHAR2(200);
  brod   NUMBER(2) := 50;
  ime VARCHAR2(15) := 'PERA';
  mbr NUMBER := 100;
  r RADNIK%ROWTYPE;
BEGIN
  sql_izraz := 'INSERT INTO  RADNIK (MATBR, LIME, BROD)
    VALUES (:1, :2, :3)';
  EXECUTE IMMEDIATE sql_izraz USING mbr, ime, brod;

  sql_izraz:= 'SELECT * FROM RADNIK WHERE MATBR = :id';
  EXECUTE IMMEDIATE sql_izraz INTO r USING 7788;

  EXECUTE IMMEDIATE 'DELETE FROM RADNIK
    WHERE BROD = :n' USING brod;
```

Dodavanje podataka o novom radniku.

Upit koja vraća podatke o konkretnom radniku.

Brisanje radnika određenog odeljenja.

Dinamički SQL

```
plsql_block := 'BEGIN PREDUZECE.povecaj_platu(:id, :pr);
    END;';
  EXECUTE IMMEDIATE plsql_blok USING brod, 500;

  EXECUTE IMMEDIATE 'CREATE TABLE bonus (id
    NUMBER, iznos NUMBER)';

  sql_izraz := 'ALTER SESSION SET SQL_TRACE
    TRUE';
  EXECUTE IMMEDIATE sql_izraz;
END;
```

Izvršavanje PL/SQL bloka koji poziva proceduru.

DDL naredba.

Administracija Oracle baze podataka.

Dinamički SQL

```
DECLARE
  TYPE RadnikCursorType IS REF CURSOR;
  radnici RadnikCursorType;
  r  RADNIK%ROWTYPE;
  sql_izraz VARCHAR2(100);
  brod   NUMBER := 50;
BEGIN
  sql_izraz := 'SELECT * FROM PREDUZECE.RADNIK
    WHERE BROD = :brojod';
  OPEN radnici FOR sql_izraz USING brod;
  LOOP
    FETCH radnici INTO r;
    EXIT WHEN radnici%NOTFOUND;
    -- obrada sloga
  END LOOP;
  CLOSE radnici;
END;
```

Za izvršavanje dinamičkih SELECT naredbi koje vraćaju više vrsta se koriste kursori.

Dinamička naredba se specificira u FOR delu naredbe za otvaranje kursora.

Sekvence

- Sekvenca je Oracle objekat koji se koristi za generisanje serija (sekvenci) numeričkih vrednosti.
- Sekvenca se koristi u situacijama kada je potrebno obezbediti automatsko generisanje vrednosti nekod identifikatora. Najčešće se koristi za potrebe vrednosti surogat ključa (autonumber, identity).

```
CREATE SEQUENCE <ime sekvence>
   [MINVALUE <minimalna vrednost> | NOMINVALUE]
   [MAXVALUE <maksimalna vrednost> | NOMAXVALUE]
   [START WITH <početna vrednost>]
   [INCREMENT BY <korak>]
   [CACHE <vrednost> | NOCACHE]
   [CYCLE | NOCYCLE]
   [ORDER | NOORDER];
```

Sekvence

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

INSERT INTO RADNIK (ID, IME, PREZIME)
VALUES(radnik_id.NEXTVAL, 'Milan ', 'Perić ');

SELECT radnik_id.NEXTVAL FROM DUAL;

SELECT radnik_id.CURRVAL FROM DUAL;
```

Kreiranje nove sekvence.

Sekvenca kao AUTONUMBER.

Inkrementira se sekvenca i vraća sledeća vrednost.

Tekuća vrednost sekvence.

Slogovi

- Slog predstavlja kolekciju srodnih podataka pri čemu svaki podatak ima svoje ime i tip.
- Podaci u slogu su logički povezani ali različitog tipa.
- Slog je sličan konceptu strukture koji postoji kod C/C++ programskog jezika.
- Atribut %ROWTYPE omogućava da se definiše slog koji predstavlja jednu vrstu tabele. Korisnik ne može da bira tipove podataka za pojedinačne kolone u ovako definisanom slogu.
- Tip podataka RECORD omogućava da se izbegnu ova ograničenja.

Slogovi

- Za deklaraciju slogova se koristi sledeća sintaksa:

TYPE <ime sloga> **IS RECORD** (<deklaracija polja>[,<deklaracija polja>]…);

- Pri tome deklaracija polja ima sledeću sintaksu:

<ime polja> <tip podatka> [[**NOT NULL**] {**:=** | **DEFAULT**}<expression>];

- Kao tip podataka mogu se koristiti standardni ORACLE i PL/SQL tipovi, korisnički definisani tipovi ili drugi slogovi.
- Nakon deklaracije slog se ponaša kao i bilo koji tip podataka koji podržavaju ORACLE i PL/SQL.
- Slogovi se ne mogu porediti sa NULL vrednostima niti se za dva sloga može proveravati da li su jednaki. Moraju se proveriti pojedinačna polja koja čine slog.

Slogovi

```
DECLARE
TYPE ImeR IS RECORD (ime VARCHAR(20),
        prezime PREDUZECE.RADNIK.PREZIME%TYPE,
        sslovo CHAR(1));
TYPE Radnik IS RECORD (ime ImeR,
                mbr INT NOT NULL := 0,
                plata NUMBER(10,2));
r ImeR;
BEGIN
      SELECT LIME, PREZIME, INIC
        INTO r.ime, r.prezime, r.sslovo
      FROM PREDUZECE.RADNIK
      WHERE MARBR = 10;

      SELECT LIME, PREZIME, INIC
        INTO r                    FROM
    PREDUZECE.RADNIK
      WHERE MARBR = 10;

      DBMS_OUTPUT.PUT_LINE(r.ime || ' ' || r.prezime);
END;
```

Deklaracija sloga. Za tipove polja mogu se koristiti eksplicitno deklarisani tipovi ili implicitno deklarisani tipovi korišćenjem atributa %TYPE I %ROWTYPE.

Za tip polja se mogu koristiti i prethodno deklarisani slogovi (nested records ili ugnježđeni slogovi).

Moguće je pristupati pojedinačnim poljima u slogu ili slogu kao celini..

Kolekcije

- PL/SQL nudi podršku za rad sa kolekcijama podataka.
- Kolekcija podataka predstavlja uređenu grupu elemenata istog tipa.
- PL/SQL kolekcije predstavljaju ekvivalent nizovima koji se javljaju kod standardnih programskih jezika.
- Kolekcije mogu da imaju samo jednu dimenziju i mogu biti indeksirane samo korišćenjem celobrojnih vrednosti.
- PL/SQL podržava dva tipa kolekcija:
  - VARRAY – nizovi promenljive dužine
  - TABLE – tabela sa samo jednom kolonom

VARRAY

- Podaci tipa VARRAY moraju da imaju definisanu gornju granicu.
- Indeksi kod promenljivih tipa VARRAY moraju biti strogo sukcesivni.
- Promenljive tipa VARRAY su pogodne za čuvanje manjih kolekcija.
- Oracle skladišti informacije o promenljivama tipa VARRAY tako da indeksi i uređenje ostaju očuvani.

VARRAY

- Za definiciju tipa VARRAY se koristi sledeća sintaksa:

```
--ukoliko se VARRAY deklariše u nekom PL/SQL bloku
TYPE <ime tipa>
IS VARRAY (<gornja granica>)
OF <tip podataka> [NOT NULL];
```

```
--ukoliko se VARRAY deklariše van PL/SQL bloku
CREATE [OR REPLACE] TYPE <ime tipa>
IS VARRAY (<gornja granica>)
OF <tip podataka> [NOT NULL];
```

- Vrednost indeksa se kreće od 1 do gornje granice.

Tabele

- Tabele ne moraju da imaju definisanu gornju granicu već se njihova dimenzija dinamički menja.
- Indeksi kod tabela ne moraju biti sukcesivni (odnosno nakon brisanja mogu da se jave šupljine u indeksima).
- Oracle skladišti tabele tako da formira posebnu sistemsku tabelu. Prilikom skladištenja ne čuvaju se redosled i indeksi podataka.
- Tabele su efikasne kada se vrši pretraživanje veće količine podataka.

Tabele

- Tabele mogu biti:
  - Ugnježdene tabele (nested tables)
  - Index-by tabele
- Ova dva tipa tabela imaju sličnu strukturu i elementima se pristupa na isit način.
- Ugnježdene tabele su naprednije jer mogu da se koriste u klasičnim SQL operacijama: SELECT; INSERT, UPDATE, DELETE.
- Ugnježdene tabele se mogu čuvati u koloni neke tabele u bazi podataka dok Index-by tabele ne mogu.
- Kod Index-by tabela indeksi elemenata mogu imati negativne vrednosti.
- Kod Index-by tabele indeksi u startu ne moraju biti sukcesivni. Kod ugnježdenih tabela indeksi u startu moraju biti sukcesivni a zatim se kasnije brisanjem ta sukcesivnost gubi.

Tabele

- Za definiciju tabela koristi se sledeća sintaksa:

```
--ukoliko se tabela deklariše u nekom PL/SQL bloku

--ugnježdena tabela
TYPE <ime tipa>
IS TABLE OF <tip podataka> [NOT NULL];

--Index-by tabela
TYPE <ime tipa>
IS TABLE OF <tip podataka> [NOT NULL]
INDEX BY BINARY_INTEGER;
```

```
--ukoliko se VARRAY deklariše van PL/SQL bloku

--ugnježdena tabela
CREATE [OR REPLACE] TYPE <ime tipa>
IS TABLE
OF <tip podataka> [NOT NULL];

--Index-by tabela
CREATE [OR REPLACE] TYPE <ime tipa>
IS TABLE (<gornja granica>)
OF <tip podataka> [NOT NULL]
INDEX BY BINARY_INTEGER;
```

Kolekcije - primeri

```
DECLARE
TYPE ImenaR IS VARRAY (20) OF VARCHAR(30);
Imena1 ImenaR;
Imena2 ImenaR := ('Mika', NULL, 'Pera' , 'Žika' );
Imena3 ImenaR;
BEGIN
Imena1 := ('Aca', NULL, NULL, NULL, 'Sima' );
Imena3 := ImenaR();
IF (Imena2(1) = 'Mika')  THEN
    DBMS_OUTPUT.PUT_LINE(Imena1(5));
END IF;
END;
```

Deklaracija tipa VARRAY.

Inicijalizacija u bloku za deklaraciju. Ne moraju se svi elementi inicijalizovati. Vrednosti nepoznatih elemenata se mogu inicijalizovati na NULL.

Inicijalizacija u telu programa.

Čak iako se promenljiva ne inicijalizuje vrednostima potrebno je pozvati prazan konstruktor. Promenljiva Imena3 ima vrednost NULL.

Kolekcije - primeri

```
DECLARE
TYPE ImenaR IS TABLE OF VARCHAR(30)
INDEX BY BINARY_INTEGER;
Imena1 ImenaR;
Imena2 ImenaR;
i BINARY_INTEGER;
BEGIN
SELECT LIME
INTO  Imena1(10)
FROM PREDUZECE.RADNIK
WHERE MATBRE = 100;

Imena2 := Imena1;

i := 10;
DBMS_OUTPUT.PUT_LINE(Imena2(i));

END;
```

Deklaracija tipa TABLE.

U Index-by tabeli indeksi ne moraju imati sukcesivne vrednosti.

Čitava kolekcija se može dodeliti drugoj promenljivoj istog tipa.

Kolekcije s ene mogu međusobno porediti.

Metode za rad sa kolekcijama

- Metode za rad sa kolekcijama se pozivaju na sledeći način:

```
<ime kolekcije>. <ime metode>[<lista argumenata>];
```

1. EXISTS – proverava da li određeni element kolekcije postoji
2. COUNT – vraća broj elemenata kolekcije
3. LIMIT – vraća gornu granicu kolekcije ukoliko ona postoji.
4. FIRST and LAST – vrćaju prvi i poslednji element kolekcije
5. PRIOR and NEXT – vraćaju prethodni i naredni element
6. EXTEND - povećava dimenzije kolekcije
7. DELETE i TRIM - uklanjaju elemente iz kolekcije

Kolekcije - primeri

```
DECLARE
TYPE ImenaT IS TABLE OF VARCHAR(30);
Imena1 ImenaT;
BEGIN

IF Imena1.FIRST = Imen1.LAST THEN ...

IF Imen1.EXISTS(3) THEN pom = Imena1(3) ...

FOR i IN 1..Imena2.COUNT LOOP ...

Imena1.TRIM;
Imena1.TRIM(3);

Imena1.DELETE(3);
Imena1.DELETE(3, 3);
Imena1.DELETE(3, 7);

i := Imen1.FIRST;
WHILE i IS NOT NULL LOOP
  -- OBRADA
  i := Imena1.NEXT(i);
END LOOP;

END;
```

- Tabela ima samo jedan element.
- Provera dali element ima validnu vrednost.
- Petlja koja obilazi sve elemente kolekcije.
- Brisanje elemenata sa kraja kolekcije.
- Brisanje elemenata iz kolekcije.
- Obilazak elemenata kolekcije.

Objekti

- Oracle objekat je korisnički definisani tip podataka koji objedinjuje podatke (atribute) zajedno sa funkcijama i procedurama (metode) koje su neophodne za manipulaciju tim podacima.
- Definisanjem objekta Oracle specificira šablon (template) koji određuje atribute i metode koje instance tog objekta moraju da imaju.
- Na osnovu specificiranog šablona se kreiraju instance tog objekta koje imaju konkretne vrednosti atributa.
- Objekti smanjuju kompleksnost sistema jer ga dele na manje logičke celine.

Objekti

- Objekti omogućavaju kreiranje softverskih komponenti koje su modularne, mogu se koristiti više puta i lakše su za održavanje.
- Definicija objkekta se sastoji iz dva dela:
  - specifikacija objekta (object specification) – definiše interfejs objekta odnosno definiše atribute i metode koje objekat ima
  - telo objekta (object body) – implementira specifikaciju odnosno potpuno definiše metode od kojih se objekat sastoji.
- Sve informacije koje su klijentskom programu neophodne za korišćenje objekta nalaze se u specifikaciji.
- Svi atributi objekta moraju biti deklarisani pre deklaracije metoda. Atributi objekta ne mogu biti deklarisani u telu objekta.

Objekti

```
CREATE [OR REPLACE] TYPE <ime tipa objekta>
  [AUTHID {CURRENT_USER | DEFINER}] {IS | AS} OBJECT (
  <ime atributa> <tip podatka> [, <ime atributa> <tip podatka>]...
  [{MAP | ORDER} MEMBER <deklaracija funkcije>,]
  [{MEMBER | STATIC} {<deklaracija potprograma> | <specifikacija poziva>}
  [, {MEMBER | STATIC} {<deklaracija potprograma> | <specifikacija poziva>}]...]
);

[CREATE [OR REPLACE] TYPE BODY <ime tipa objekta> {IS | AS}
  { {MAP | ORDER} MEMBER <telo funkcije>;
   | {MEMBER | STATIC} {<telo funkcije> | <specifikacija poziva>};}
  [{MEMBER | STATIC} {<telo funkcije> | <specifikacija poziva>};]...
END;]
```

Objekti

- Oracle objekat sadrži deklaracije atribute i metode. Objekat ne može da sadrži deklaracije konstanti, izuzetaka, kursora ili drugih tipova.
- Svi elementi objekta su javni tako da im se bez ograničenja može pristupati iz klijentskih aplikacije.
- Oracle objekat obavezno mora da sadrži bar jedan atribut a maksimalan broj atributa je 1000. Nakon definisanja objekta nije moguće dodavati nove atribute.
- Za svaki atribut se definiše ime i tip podataka. Ime atributa mora biti jedinstveno unutar objekta.
- Ne mogu se iskoristiti sledeći tipovi podataka: LONG, LONG RAW, NCLOB, NCHAR, NVARCHAR2, PL/SQL specifični tipovi podataka i tipovi podataka koji su deklarisani u drugim PL/SQL blokovima.
- Prilikom deklaracije atributa nije moguće izvršiti njihovu inicijalizaciju, nije moguće koristiti DEFAULT vrednosti i nije moguće iskoristiti NOT NULL ograničenja.
- Prilikom deklaracije atribut amoguće je koristiti prethodno deklarisane objekte (ugnježdeni objekti).

Objekti

- Metod predstavlja potprogram deklarisan unutar objekta.
- Metod može imati ime koje je identično imenu objekta ili imenu nekog od atributa unutar objekta.
- Metodi koji su deklarisani korišćenjem ključne reči MEMBER se pozivaju kod instanci objekta. Metodi koji su deklarisani korišćenjem ključne reči STATIC se pozivaju korišćenjem tipa objekta.
- Svaki metod koji je deklarisan u specifikaciji objekta mora da ima odgovarajuću implementaciju u telu objekta.
- MEMBER metode mogu da koriste ugrađeni SELF parametar koji referencira tekuću instancu objekta. STATIC metode ne mogu da referenciraju SELF parametar.
- Svaki objekat ima sistemski definisani konstruktor koji je metoda koja ima isto ime kao i objekat. Parametri konstruktora su svi atributi objekta. Konstruktor se koristi za kreiranje i inicijalizaciju instance objekta.

Objekti

```
CREATE TYPE Complex AS OBJECT
(
  rpart REAL,
  ipart REAL,
  MEMBER FUNCTION plus (x Complex) RETURN Complex,
  MEMBER FUNCTION less (x Complex) RETURN Complex,
  MEMBER FUNCTION times (x Complex) RETURN Complex,
  MEMBER FUNCTION divby (x Complex) RETURN Complex
);
```

```
CREATE TYPE BODY Complex AS
  MEMBER FUNCTION plus (x Complex) RETURN Complex IS
  BEGIN
    RETURN Complex(rpart + x.rpart, ipart + x.ipart);
  END plus;

  MEMBER FUNCTION less (x Complex) RETURN Complex IS
  BEGIN
    RETURN Complex(rpart - x.rpart, ipart - x.ipart);
  END less;

  MEMBER FUNCTION times (x Complex) RETURN Complex IS
  BEGIN
    RETURN Complex(rpart * x.rpart - ipart * x.ipart,
                   rpart * x.ipart + ipart * x.rpart);
  END times;

  MEMBER FUNCTION divby (x Complex) RETURN Complex IS
    z REAL := x.rpart**2 + x.ipart**2;
  BEGIN
    RETURN Complex((rpart * x.rpart + ipart * x.ipart) / z,
                   (ipart * x.rpart - rpart * x.ipart) / z);
  END divby;
END;
```

Objekti

```
CREATE TYPE Rational AS OBJECT (
  num INTEGER,
  den INTEGER,
  MEMBER PROCEDURE normalize;
  ...
);

CREATE TYPE BODY Rational AS
  MEMBER PROCEDURE normalize IS
    g INTEGER;
  BEGIN
    g := gcd(SELF.num, SELF.den);
    g := gcd(num, den);
    num := num / g;
    den := den / g;
  END normalize;
  ...
END;
```

Atributi objekta unutar njegovih metoda mogu se referencirati korišćenjem paramtera SELF ali i bez njega..

Objekti

```
CREATE TYPE Rational AS OBJECT (
  num INTEGER,
  den INTEGER,
  MAP MEMBER FUNCTION convert RETURN REAL,
  ...
);

CREATE TYPE BODY Rational AS
  MAP MEMBER FUNCTION convert RETURN REAL IS
  BEGIN
    RETURN num / den;
  END convert;
  ...
END;
```

Funkcije ORDER ili MAP se koriste za poređenje objekata. Ovim funkcijama se definiše uređenost objekata za potrebe operatora poređenja, DISTINCT, GROUP BY i ORDER BY funkcija.

Objekat može da ima samo jednu funkciju ovog tipa.

```
CREATE TYPE Customer AS OBJECT (
  id   NUMBER,
  name VARCHAR2(20),
  addr VARCHAR2(30),
  ORDER MEMBER FUNCTION match (c Customer) RETURN
       INTEGER
);

CREATE TYPE BODY Customer AS
  ORDER MEMBER FUNCTION match (c Customer) RETURN
       INTEGER IS
  BEGIN
    IF id < c.id THEN
      RETURN -1;
    ELSIF id > c.id THEN
      RETURN 1;
     ELSE
      RETURN 0;
    END IF;
  END;
END;
```

Objekti

```
DECLARE
  r Rational;
  r1 Rational := Rational(1, 2);
  PROCEDURE calc (numb IN OUT Rational) IS ...
  FUNCTION calc1 (num1 IN Rational, num2 IN Rational)
    RETURN Rational IS ...
BEGIN
  r := Rational(6, 8);

  r := Rational(den => 6, num => 5);

  r := calc1(Rational(8, 6), Rational(9, 2));

  r.normalize;
  r.normalize().reciprocal();

  DBMS_OUTPUT.PUT_LINE(r.num);
```

Ukoliko se eksplicitno ne pozove konstruktor objekta prilikom deklaracije objekat se inicijalizuje na NULL vrednost.

Objekti se mogu koristiti kao argumenti funkcija i procedura.

Mogu se koristiti i kao povratne vrednosti funkcija.

Pozivanje metoda objekata.

Objekti

```
CREATE TABLE numbers (rn Rational, ...);

INSERT INTO numbers (rn) VALUES (Rational(3, 62));

SELECT n.rn.num INTO my_num FROM numbers n ... ;

UPDATE numbers n SET n.rn = n.rn.reciprocal ...;
```

Objekti se mogu koristiti za definisanje kolona u tabelama.

Objekti

```
CREATE TYPE Person AS OBJECT (
  first_name   VARCHAR2(15),
  last_name    VARCHAR2(15),
  birthday     DATE,
  home_address Address,
  phone_number VARCHAR2(15));

CREATE TABLE persons OF Person;

DECLARE
  p1 Person;
  p2 Person;
BEGIN

 INSERT INTO employees
 SELECT * FROM persons p WHERE p.last_name LIKE '%Smith';

 SELECT VALUE(p) INTO p1 FROM persons p
 WHERE p.last_name = 'Kroll';
  p2 := p1;
...
END;
```

Object tables se koriste da pamte informacije o objektima. Svaka kolona tabele odgovara jednom atributu objekta.

Svakom objektu se dodeljuje object identifier koji se koristi za jednoznačnu identifikaciju objekata u tabeli..

Podupit vraća vrste čije kolone odgovaraju atributima objekata.

Operator VALUE obezbeđuje da se umesto rezultujuće vrsta dobije objekat .