Capitolul 13. Obiecte în Oracle

În acest capitol sunt prezentate modalitățile prin care putem defini și utiliza obiecte în Oracle, obiectele reprezentând o soluție modernă de organizare și manipulare a datelor complexe.

13.1 Definirea unui obiect

În Oracle un tip de obiecte (sau o clasă în limbajul C++ sau Java) se declară folosind comanda pentru crearea unui tip de date definit de utilizator (CREATE TYPE) [22]. Utilizarea obiectelor prezintă numeroase avantaje cum ar fi:

- Permit stocarea de date complexe, imagini, secvențe audio și video;
- Oferă un nivel înalt de organizare și abstractizare a datelor;
- Sunt reutilizabile, aceasta ducând la o productivitate crescută în programare;
- Pot încapsula operații în același timp cu date [22]. De exemplu, un obiect de tip ordin de plată poate conține și metode de calcul a costurilor;

Oracle este un sistem de gestiune a bazelor de date obiect-relațional (SGBDOR), adică capacitatea de a defini și manipula obiecte vine ca o extensie a modelului relațional clasic – foarte popular, astfel permițând toate operațiile complexe de interogare oferite de modelul relațional și în același timp oferind avantajele manipulării datelor complexe abstractizate sub forma obiectelor.

Un tip de obiect (o clasă) este de fapt un tip de date [22]. Se poate declara ca orice tip de date în Oracle, putem declara coloane din tabel de tip obiect, variabile de tip obiect etc. Un obiect, adică o valoare a variabilei, este o instanță a respectivului tip de obiect (Figura 13.1).

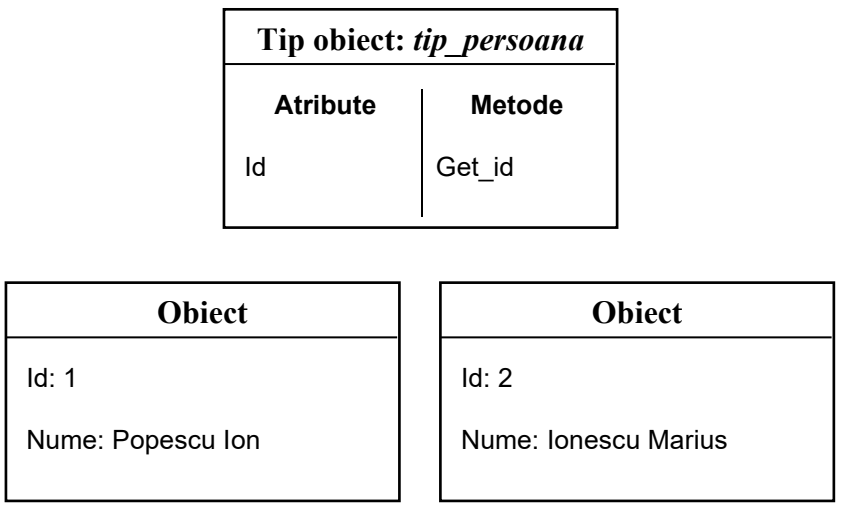


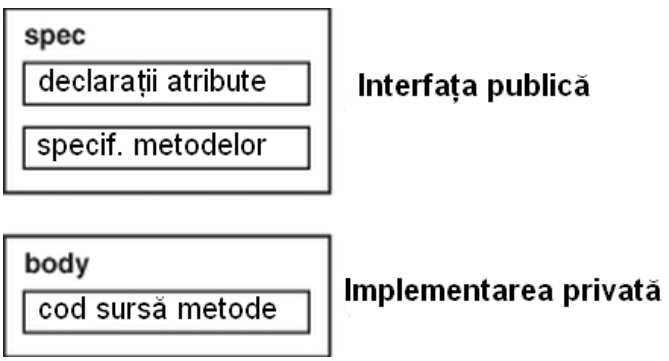
Figura 13.1 Un tip obiect și instanțe ale acestuia [22]

Exemplu:

Comanda SQL următoare declară un tip de obiect numit „tip_persoana”. Atributele sunt: id, nume și telefon, iar metodele ‚get_id’ și ‚afiseaza’.

```
CREATE TYPE tip_persoana AS OBJECT (
  id          NUMBER,
  nume          VARCHAR2(30),
  telefon         VARCHAR2(20),
  MAP MEMBER FUNCTION get_id RETURN NUMBER,
  MEMBER PROCEDURE afiseaza (
        SELF IN OUT NOCOPY tip_persoana)
 );
/
CREATE TYPE BODY tip_persoana AS
  MAP MEMBER FUNCTION get_id RETURN NUMBER IS
  BEGIN
    RETURN id;
  END;
  MEMBER PROCEDURE afiseaza (
         SELF IN OUT NOCOPY tip_persoana ) IS
          BEGIN
            DBMS_OUTPUT.PUT_LINE('Numele e' || ' ' || nume);
          DBMS_OUTPUT.PUT_LINE('Telefonul e' || ', ' || telefon);
          END;
END;
/
```

Comanda CREATE TYPE BODY detaliază metodele obiectului ‚tip_persoana'. Metodele conțin operații efectuate de obiecte (așa numitul comportament). Comenzile anterioare nu alocă și spațiu pentru obiecte [22].



În exemplele următoare vom crea instanțe ale clasei ‚tip_persoana', adică mai multe obiecte. Pentru aceasta vom crea o tabelă ‚contacte' care conține o coloană de tip obiect, iar apoi se pot insera mai multe linii în această tabelă.

```
CREATE TABLE contacte (
  contact         tip_persoana,
  contact_date    DATE );

INSERT INTO contacte VALUES (
  tip_persoana (1, 'Ion Popescu', '1234'), '05 Dec 2019' );
```

Prin metodele obiectelor putem avea acces la valorile atributelor, modifica aceste valori, sau efectua anumite operații. În continuare putem observa un exemplu de apelare a metodei get_id, pentru a afișa ID-urile persoanelor din tabela Contacte.

O categorie specială de metode sunt ‚constructorii'. Constructorul este folosit la crearea și inițializarea instanțelor unei clase (crearea unui obiect) și au același nume cu al clasei.

```
SELECT c.contact.get_id () FROM contacte c;
```

Putem de asemenea inițializa obiecte în cadrul unui program PL/SQL. În exemplul următor se declară un obiect ‚p' de tip ‚tip_persoana', care este apoi inițializat prin metoda constructor ‚tip_persoana'. Apoi se apelează metoda ‚afiseaza'.

```
set serveroutput on;
DECLARE
   p Tip_persoana;
BEGIN
   p := Tip_Persoana(1, 'Ion Popescu', '1234');
   p.afiseaza;

END;
/
```

Există și varianta de a crea o tabelă de obiecte, ca în exemplul următor.

```
CREATE TABLE tabela_persoane OF tip_persoana;

INSERT INTO tabela_persoane VALUES (1, 'Florin Marinescu', '745');

SELECT * FROM tabela_persoane p
       WHERE p.nume = 'Florin Marinescu';
```

Exerciții:

- Inserați mai multe linii in tabelele de obiecte
- Creați un obiect de tip dreptunghi, cu atributele lățime și lungime, și o funcție care calculează aria.

13.2 Crearea unei referințe la un obiect (REF)

Având o tabelă de obiecte, în care fiecare linie este un obiect (row object), tabelă precum „tabela_persoane" de mai sus, se pot defini referințe la obiecte prin specificatorul REF.

Exemplu – definim un tip de obiect care conține un atribut de tip REF, adică o referință la un obiect [22].

```
CREATE TYPE tip_persoana2 AS OBJECT (
 nume     VARCHAR2(30),
 manager  REF tip_persoana2 );
 /
```

Apoi, prin comanda următoare, creăm o tabelă de obiecte.

```
CREATE TABLE tabela_pers2 OF tip_persoana2;
```

Și inserăm o linie, care are valoarea NULL pentru coloana ‚manager'.

```
INSERT INTO tabela_pers2 VALUES ( tip_persoana2('Vasile Ion', NULL));
```

Apoi, mai adăugăm o linie cu un angajat cu numele ‚Mihai Popescu', care este subordonat managerului ‚Vasile Ion':

```
INSERT INTO tabela_pers2
 SELECT tip_persoana2('Mihai Popescu', REF(t))
     FROM tabela_pers2 t
     WHERE t.nume = 'Vasile Ion';
```

Putem afișa simplu conținutul tabelei:

```
SELECT * FROM Tabela_pers2;
```

Sau, pentru a vedea subordonările:

```
SELECT t.nume "nume angajat" , t.manager.nume "nume manager"
            FROM Tabela_pers2  t
            WHERE t.nume = 'Mihai Popescu';
```

Putem afișa conținutul tabelei și prin meniurile SQL Developer, și obținem un rezultat precum în figura următoare.

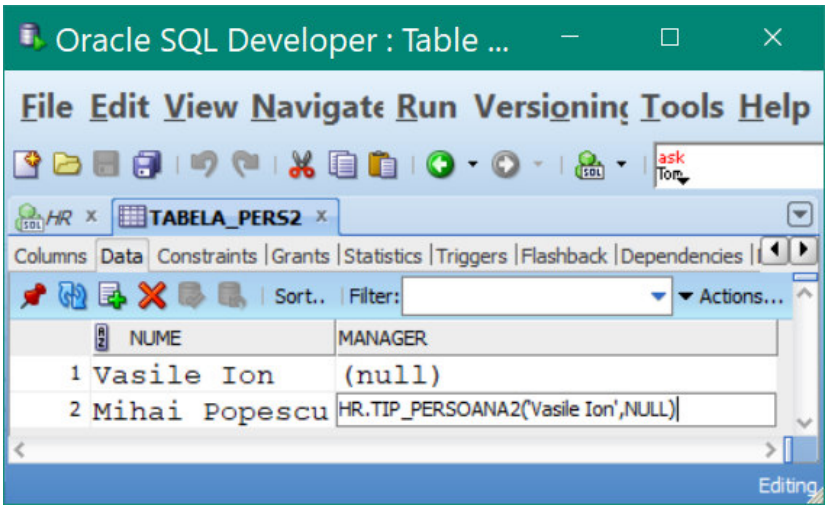


Figura 13.2 Afișarea conținutului tabelei Tabela_pers2

13.3 Crearea de subtipuri de obiecte

Moștenirea tipurilor de obiecte permite crearea de ierarhii de tipuri de obiecte, adică niveluri succesive de subtipuri care specializează supertipuri de obiecte ([22]). Obiectele -subtip- moștenesc atributele și metodele obiectelor -supertip-, și pot fi adăugate noi atribute și metode, sau pot fi suprascrise metode ale supertipului.

Exemplu – crearea unui tip de obiect supertip numit Vehicul, cu atributele nr (număr de înmatriculare), marcă și preț, și metodele get_nr și afișează:

```
CREATE OR REPLACE TYPE Vehicul AS OBJECT (
  nr          VARCHAR(10),
  marca       VARCHAR2(30),
  pret        NUMBER(5),
  MAP MEMBER FUNCTION get_nr RETURN NUMBER,
  MEMBER FUNCTION afiseaza RETURN VARCHAR2)
  NOT FINAL;
/
```

Și detalierea metodelor clasei Vehicul:

```
CREATE OR REPLACE TYPE BODY Vehicul AS
   MAP MEMBER FUNCTION get_nr RETURN NUMBER IS
     BEGIN
         RETURN nr;
     END;
    -- functie care poate fi suprascrisa de subtipuri
```

```
    MEMBER FUNCTION afiseaza RETURN VARCHAR2 IS
        BEGIN
            RETURN 'Nr inmatriculare: ' || TO_CHAR(nr) || ', Marca: ' || marca || ', Pret: '
|| pret;
         END;
END;
/
```

Apoi, în exemplul următor, creăm un subtip al tipului de obiect Vehicul, numit Automobil, care va moșteni toate atributele și metodele supertipului, și va avea în plus și două atribute specifice (nr_pasageri și viteza_max).

```
CREATE TYPE Automobil UNDER Vehicul (
     nr_pasageri NUMBER,
     viteza_max NUMBER,
     OVERRIDING MEMBER FUNCTION afiseaza RETURN VARCHAR2)
     NOT FINAL;
/
```

La detalierea metodelor specificăm doar că pentru subtip funcția ‚afiseaza' este suprascrisă (modificată).

```
CREATE TYPE BODY Automobil AS
    OVERRIDING MEMBER FUNCTION afiseaza RETURN VARCHAR2 IS
    BEGIN
       RETURN (self AS Vehicul).afiseaza || ' - Nr pasageri: ' || nr_pasageri ;
    END;
END;
/
```

Putem testa apoi funcționarea obiectelor – subtip prin comenzile următoare:

```
CREATE TABLE autoturisme OF Automobil;

INSERT INTO autoturisme VALUES ('B-01-ABC', 'Dacia', 10000,5,180);

SELECT * FROM autoturisme;

SELECT a.afiseaza() FROM autoturisme a where a.marca='Dacia';
```

Observați funcționarea metodei ‚afiseaza' specifică subtipului.

Dacă se dorește apelarea metodei supertipului pornind de la metoda subtipului, acest mecanism se numește „invocare generalizată" (generalized invocation) [23], un exemplu fiind codul următor.

```
SET SERVEROUTPUT ON;
DECLARE
  masina  Automobil := Automobil('PH-123-AAA', 'Fiat', 5000, 5, 150);
  un_sir  VARCHAR2(100);
BEGIN
  un_sir := (masina AS Vehicul).afiseaza; -- invocare generalizata
  DBMS_OUTPUT.Put_line(un_sir);
END;
/
```

Mai sus am văzut cum funcționează mecanismul de suprascriere a metodelor (overriding). Mai există și opțiunea ‚overloading')[23], care nu trebuie specificată la definirea metodelor subtipului, ci se activează automat dacă subtipul definește o metodă cu exact același nume cu una din metodele supertipului. Compilatorul va determina apoi care din metode să apeleze în funcție de numărul și tipul de date al parametrilor metodelor.