Relational-obiectual în Oracle9i2. O noua abordare în modelarea datelor

Asist. Liviu-Gabriel CRETU
Catedra de Informatica Economica, Universitatea "Al. I. Cuza" Iasi

We are all aware that object-oriented technology has brought some tremendous advantages in modern software-applications development. Yet, in database's world the relational model is still leading the way. This paper tries to emphasize a new perspective of data modeling in database systems offered by the latest Oracle's RDMS product-Oracle 9i2.
Keywords: *abstract data types, objects, inheritance, polymorphism.*

Introducere

Desi în orice aplicatie informatica de întreprindere (si nu numai) procesul de modelare a datelor urmeaza, în egala masura, doua paradigme fundamentale, modelul relational si cel orientat-obiect, în ceea ce priveste persistenta datelor multa vreme modelul relational a condus detasat fiind implementat de mai toate SGBD-urile. Aceasta tendinta s-a manifestat si pe fondul aparitiei unui limbaj universal de gestionare (definire, interogare si actualizare) a datelor organizate dupa schema relationala, Structured Query Language.

Oracle ofera în prezent suport deplin pentru tehnologia orientata-obiect sub forma unui nivel de abstractizare construit deasupra modelului relational. Tipuri abstracte noi (numite si tipuri definite de utilizator) pot fi construite fie pornind de la tipurile de date scalare standard, fie pe baza unor alte tipuri abstracte, referinte la alte obiecte sau tipuri-colectii. Metadatele referitoare la tipurile abstracte sunt stocate în schema bazei de date fiind astfel disponibile în SQL, PL/SQL, Java sau alte limbaje. În spatele acestui model obiectual datele sunt în continuare stocate sub forma de tabele si atribute, dar pot fi tratate si ca entitati complexe (obiecte) din lumea reala. Utilizarea tipurilor abstracte pentru modelarea datelor poate oferi urmatoarele avantaje majore:

- Pot fi încapsulate operatii alaturi de date. Daca tabelele pot stoca doar date, tipurile abstracte ofera posibilitatea definirii unor functii (denumite metode) ce pot fi aplicate asupra datelor obiectelor. De exemplu pentru o factura, privita ca obiect ce cuprinde toate datele unui document real, se poate defini o metoda care sa calculeze suma totala si TVA-ul pentru produsele componente.
- Obiectele ofera eficienta sporita în dezvoltarea aplicatiilor datorita faptului ca sunt stocate în baza de date, includ operatiile ce se pot efectua asupra lor si pot fi accesate de orice alt modul-program. Ca urmare, programatorii nu sunt nevoiti sa recreeze structuri de mapare a datelor în fiecare aplicatie-client.
- Obiectele ofera o perspectiva de "întreg" asupra datelor (partile componente), în mod similar modului de abordare a lucrurilor din perspectiva umana. Ca urmare sunt mai usor de reprezentat si de manipulat.

Tipuri abstracte de date. Atribute si metode.

În Oracle, pentru a defini un nou tip utilizam instructiunea `CREATE TYPE` astfel:

```
CREATE TYPE numeTip AS OBJECT(<atrib_1 Tip>,
…, <atrib_n Tip>,
                [ MEMBER FUNCTION Metoda1 [(p1
tip,…)] RETURN Tip],
                [  MEMBER  PROCEDURE  Metoda2
[(p1 tip, …)]
                                          )
```

Pentru definirea tipului fiecarui membru-atribut putem apela fie la tipurile predefinite (`Number, Varchar2, Integer, etc…`) fie la alte tipuri abstracte definite anterior.

Definirea comportamentului obiectelor (operatiile pe care le pot realiza acestea asupra datelor proprii) se realizeaza prin implementarea functiilor-membru specificate la crearea tipului, astfel:

```
CREATE TYPE BODY numeTip AS
        MEMBER FUNCTION Metoda1 IS
        … <variabile locale>…
        BEGIN
                … Corpul metodei…
```

```
        END Metoda1;
…
END;
```

Un obiect nou de un anumit tip se obtine cu ajutorul constructorului implicit (o metoda cu acelasi nume ca si al tipului si parametri pentru toate atributele declarate):

`NEW DenumireTip(val1, val2, .. valn)`, unde `valx` reprezinta o valoare pentru fiecare atribut al tipului

Un aspect deosebit de important în ceea ce priveste manipularea obiectelor într-un limbaj de programare se refera la modul de transfer al acestora (între variabile sau între un modul-program si altul). În acest sens este important de retinut faptul ca *în Oracle obiectele se transmit în mod implicit prin valoare si nu prin referinta* (asa cum se întâmpla de exemplu în Java). Astfel, fiind date doua obiecte, obj1 si obj2, adresate prin doua variabile, v1 respectiv v2, în urma atribuirii v1:=v2 nu vom obtine doua referinte spre un acelasi obiect (obj2) ci vom dispune de doua variabile ce puncteaza spre doua obiecte complet distincte dar cu un continut identic.

Stocarea si gestionarea obiectelor în baza de date

Oracle furnizeaza doua tehnologii de stocare a obiectelor în baza de date, tehnologii ce au determinat utilizarea a doua sintagme diferite:

- ***obiecte linie*** – sunt stocate sub forma de linii ale unei tabele construita pe baza tipului caruia îi apartin respectivele obiecte; atributele unei asemenea tabele vor fi tocmai atributele tipului iar un obiect va constitui o înregistrare;
- ***obiecte coloana*** – persistenta se asigura sub forma de valori ale unui atribut declarat într-o tabela relationala clasica; pentru fiecare înregistrare, respectivul atribut va contine un obiect de tipul declarat.

Listingul 1 prezinta cele doua modele de persistenta a obiectelor în baza de date si evidentiaza faptul ca tabelele obiectuale (`Test_ObjTbl`) pot fi tratate ca si tabelele relationale în ceea ce priveste actualizarea sau interogarea datelor stocate de atributele obiectelor. Spre deosebire de tabelele obiectuale, pentru actualizarea si interogarea datelor tabelelor relational-obiectuale (`Test_RelTbl`) este obligatorie utilizarea unui alias pentru tabela în cauza pentru a urma apoi schema: `AliasTabela.AtributTabela.AtributObiect` (altfel obtinem eroarea `ORA-00904 – invalid identifier`).

Listing 1. Doua modalitati de stocare a obiectelor în BD

```
CREATE TYPE TEST AS OBJECT (a1 Integer, a2 Char(1))
/
CREATE TABLE Test_ObjTbl OF TEST;
                -- sau :
CREATE TABLE Test_RelTbl ( id Integer, test_col TEST);
-- adaugam obiecte ca în orice alta tabela relationala:
INSERT INTO Test_ObjTbl VALUES (111,'A');
                --sau prin crearea în mod explicit a unui obiect nou:
INSERT INTO Test_ObjTbl values (new  Test(112,'B' )) ;
                -- inserare date si obiecte in tabela relational-obiectula:
        INSERT INTO Test_RelTbl VALUES (1,new Test(111,'A' ));
INSERT INTO Test_RelTbl VALUES (2, new Test (112,'B'));
-- interogarea si actualziarea datelor obiectelor din tabela obiectuala
SELECT a1 FROM Test_ObjTbl;
UPDATE Test_ObjTbl SET a1=116 WHERE a1=112;
-- interogarea si actualizarea datelor obiectelor din tabela relationala:
SELECT T.test_col.a1 FROM Test_RelTbl T;
UPDATE Test_RelTbl T SET T.test_col.a1=116 WHERE T.test_col.a1=112
```

În Listing 1 s-au prezentat tehnici de actualizare si interogare a ***datelor*** obiectelor (valorile stocate de atribute). De multe ori este însa necesar ca în urma unei interogari sa obtinem chiar ***obiecte*** sau sa înlocuim în tabela obiectuala un obiect cu altul complet nou. Pentru realizarea acestui deziderat utilizam functia `VALUE(alias_tabela_ob)` care returneaza *obiecte noi*, identice cu obiectele din tabela-obiectuala specificata prin `alias_tabela_ob`. Functia poate fi utilizata numai într-o fraza SQL si numai pentru tabele obiectuale. Bib-

cul PL/SQL urmator initializeaza variabila _v cu un obiect din tabela `Test_ObjTbl`.

Referinte spre obiecte

Asa cum precizam si ceva mai devreme, în orice aplicatie orientata-obiect exista diverse situatii în care avem nevoie de o adresa logica (asa numita "pointer") spre un obiect existent, adresa prin intermediul careia sa gestionam exact obiectul respectiv si nu o copie a acestuia. Mai mult, daca privim lucrurile din perspectiva arhitecturii bazei de date, pentru tabelele copil avem nevoie de referinte spre obiectele din tabelele parinte.

În sprijinul acestor necesitati Oracle furnizeaza tipul de data REF ce defineste un *pointer spre un obiect-linie* existent, prin intermediul caruia putem modela relatii între obiecte (în special relatii de tip "one-to-many") si care poate fi utilizat în diverse scopuri: pentru a examina si actualiza datele obiectului sursa; pentru a obtine o copie a obiectului sursa; pentru a schimba obiectul sursa spre care tine pointer-ul.

Exista câteva aspecte esentiale în ceea ce priveste tipurile REF:

- o referinta reprezinta adresa, identificatorul unic (OID-Object IDentifier) generat de sistem la crearea obiectului (acest identificator este atribut automat la momentul initializarii obiectului si este unic în baza de date);
- prin intermediul referintelor manipulam obiectele în mod clasic orientat-obiect, folosind notatia cu punct;
- nu pot fi utilizate la definirea cheii primare;
- un REF poate defini un pointer numai spre un obiect stocat într-o tabela de obiecte si nu spre obiecte rezidente pe câmpurile unei tabele relationale clasice sau în memorie. Tipul obiectului trebuie sa fie obligatoriu cel declarat la definirea tipului REF , sau un tip derivat al acestuia.

Pentru exemplificare, vom defini tabela `Test_RefTbl` pentru care atributul `ref_objTest` va stoca referinte spre obiecte de tip `TEST` numai din tabela `Test_ObjTbl` (restrictie implementata prin cheie straina în stilul relational clasic), astfel:

```
CREATE TABLE Test_RefTbl (
ref_objTest  REF TEST REFERENCES Test_ObjTbl,
   alt_atribut Integer
);
```

Obtinerea unei referinte spre un obiect se realizeaza cu ajutorul functiei `REF(obiect)` care poate fi apelata doar într-o fraza SQL. Ca urmare, pentru a adauga înregistrari în tabela `Test_RefTbl` va fi necesara o subinterogare, astfel:

```
INSERT INTO Test_RefTbl
(SELECT REF(T), 1001 FROM Test_ObjTbl T WHERE
T.a1=116);
```

Într-o interogare ce utilizeaza referinte trebuie obligatoriu definit un alias pentru tabela:

```
SELECT Tr.ref_objTest.a2 from Test_RefTbl Tr
where Tr.ref_objTest.a1=116
```

Posibilitatea obtinerii unei *referinte* spre un obiect deja existent are o deosibita importanta în ceea ce priveste arhitectura codului unei aplicatii orientate-obiect din perspectiva urmatoarelor aspecte:

- o referinta spre un obiect existent ofera posibilitatea manipularii acestuia de catre mai multe module-program în functie de necesitati;
- o eventuala modificare a starii obiectului s-ar efectua într-un singur loc si ar fi imediat disponibila tuturor modulelor ce fac referire la acesta;
- referintele ofera o modalitate de gestionare mult mai riguroasa a memoriei disponibile.

PL/SQL nu suporta, la momentul actual (versiunea Oracle9i2), navigare între obiecte direct prin intermediul referintelor, desi se pot declara variabile de tip REF care sa contina adrese de obiecte. Exista totusi o posibilitate de gestionare a referintelor în PL/SQL prin intermediul pachetului `UTL_REF`.

Pentru fiecare din proceduri putem construi o fraza SQL echivalenta. Avantajul utilizarii pachetului `UTL_REF` consta în faptul ca, odata ce obtinem o referinta putem manipula obiectul fara a sti exact tabela în care se afla.

Extinderea tipurilor. Mostenire si polimorfism

Un concept esential al metodologiei orientate-obiect este reprezentat de *mostenire*, ca find procesul de specializare a tipurilor prin crearea unor *subtipuri* derivate din cel de baza, numit *supertip*. Subtipurile *mostenesc* toti membrii supertipului (date si metode) si adauga un element nou pentru a evidentia o

anumita particularitate a obiectelor (entitatile din lumea reala). Elementul de noutate se poate concretiza în:

- atribute si/sau metode noi;
- modificarea comportamentului (o alta implementare a metodelor mostenite).

Ansamblul alcatuit din tipul de baza (cel mai abstract) si toate subtipurile sale specializate este numit *ierarhie*. Figura 1 ilustreaza o posibila ierarhie a persoanelor ce-si desfasoara activitatea în mediul universitar. Specializarea subtipului poate însemna: adaugarea unor noi membri (atribute, metode) sau modificarea comportamentului uneia din metodele mostenite (proces numit *suprascriere*). Suprascrierea metodelor face posibila manifestarea *polimorfismului* (doua obiecte de acelasi supertip dar de subtipuri diferite se comporta diferit la executia metodei suprascrise).

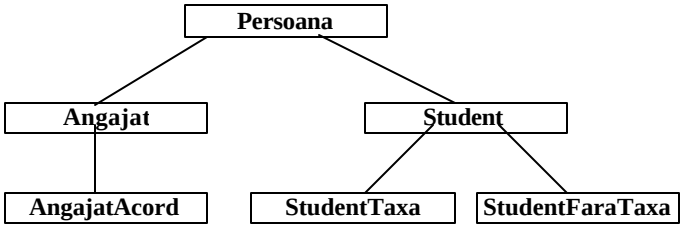

Fig. 1. O ierarhie de tipuri.

Începând cu versiunea 9i2, Oracle ofera suport deplin pentru toate conceptele descrise anterior. Astfel, definirea unui subtip derivat dintr-un supertip poate fi realizata numai daca acesta din urma a fost declarat `NOT FINAL`. De cele mai multe ori o aplicatie nu "lucreaza" cu obiecte instantiate din supertip ci cu obiecte specializate. Supertipurile se utilizeaza pentru a trata la un moment dat toate obiectele din tipurile derivate în mod unitar. Astfel, conform ierarhiei definite în figura 1, nu vor exista cazuri în care vom utiliza obiecte de tip `Persoana` sau `Student` ci doar obiecte de tip `Angajat`, `AngajatAcord`, `StudentTaxa` si `StudentFaraTaxa`. Pentru a asigura însa flexibilitatea ulterioara a aplicatiei si pentru a suporta o eventuala expansiune a modelului în viitor, aceste obiecte vor fi *tratate* în anumite faze ale prelucrarilor ca fiind de tipul de baza, Persoana. În aceeasi ordine de idei, la nivelul supertipului se pot defini metode care nu implementeaza nici un comportament datorita faptului ca acest comportament va fi diferit pentru fiecare subtip în parte. Ca urmare, proiectantul supertipului va delega responsabilitatea definirii actiunii de executat catre proiectantii subtipurilor care vor avea în vedere cerintele proprii. De exemplu, înscrierea unui student la un examen presupune verificarea îndeplinirii unor conditii prealabile diferite pentru studentii cu taxa fata de cei fara taxa (primii trebuie sa-si plateasca taxa de scolarizare în întregime pentru a putea participa la examen). Iata de ce proiectantul tipului Student va putea recurge la definirea unei eventuale metode `inscrieEx()` ca fiind abstracta, urmând ca subtipurile `StudentTaxa` si `StudentFaraTaxa` sa implementeze fiecare propriul mod de evaluare a conditiilor de intrare în examen.

Având în vedere aspectele discutate anterior, paradigma orientata-obiect ofera posibilitatea crearii unor tipuri ce nu pot fi instantiate (nu putem obtine obiecte de tipul respectiv) si a unor metode fara implementare ce vor trebui obligatoriu suprascrise de tipurile derivate.

Exemplu de implementare a unui model relational-obiectual

Pentru a ilustra avantajele aduse de tehnologia orientata-obiect în modelarea datelor, vom apela la un exemplu arhicunoscut: gestionarea rezultatelor evaluarilor studentilor unei facultati. Desi pare destul de simpla la prima vedere, problema se complica daca avem în vedere urmatoarele aspecte: în baza de date trebuie stocate notele partiale (proiect, laborator, examen final) iar nota finala trebuie calculata conform unui algoritm anume; algoritmul de calcul difera de la o disciplina la alta precum si numarul de note

ce intra în compozitia notei finale; ponderea notelor partiale în nota finala difera de la o disciplina la alta si poate chiar sa difere pentru aceeasi disciplina dar la specializari diferite (aceasta ultima varianta nu o vom lua însa in calcul din ratiuni de simplificare). Asadar, va trebui sa definim un model de date suficient de flexibil pentru a suporta extensibilitate (aparitia altor tehnici de evaluare) în timp.

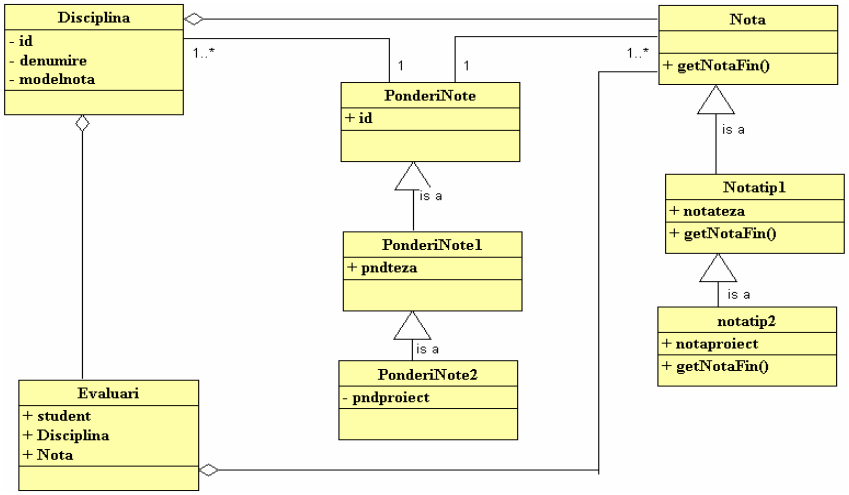

Fig. 2. Diagrama de tipuri pentru gestionarea notelor studentilor

Diagrama din figura 2 prezinta un model de date prin care se încearca rezolvarea problemei expuse ceva mai devreme, astfel:

- tipul `Nota` este definit abstract (nu poate fi instantiat), astfel ca metoda `getNotaFin()` va trebui implementata prin suprascriere de toate tipurile de note (aceasta metoda va fi apelata în scopul obtinerii notei finale);
- tipul Disciplina defineste un membru `modelNota` ce va stoca un obiect-nota de un anumit tip si care va fi utilizat pentru a identifica tipul de nota pe care îl accepta o disciplina
- un obiect de tip `Disciplina` va "tine" o referinta spre un obiect de tip `Ponderi`, referinta pe care o va prelua si un obiect-nota corespunzator disciplinei; obiectul `Ponderi` va fi utilizat în algoritmul de calcul al notei finale.

Listing 2. Script pentru implementarea modelului din figura 2 în Oracle 9i2

```
CREATE OR REPLACE TYPE PonderiNote AS OBJECT( id Integer)
NOT INSTANTIABLE NOT FINAL
/
CREATE OR REPLACE TYPE PonderiNote1 UNDER PONDERINOTE(
pndTeza number(3,2)
) NOT FINAL
/
CREATE OR REPLACE TYPE ponderiNote2 UNDER PONDERINOTE1(
pndProiect number(3,2)
) NOT FINAL
/
/* ponderile vor fi stocate într-o tabela obiectuala */
CREATE TABLE Ponderitbl OF PonderiNote (id Primary Key) ;
INSERT INTO ponderitbl VALUES (new PonderiNote1(1,1));
INSERT INTO ponderitbl VALUES (new PonderiNote2(2,0.7,0.3));
INSERT INTO ponderitbl VALUES (new PonderiNote2(3,0.5,0.5));
CREATE OR REPLACE TYPE Nota AS OBJECT (
       ponderi REF PonderiNote,
       MEMBER FUNCTION getNotaFin  RETURN number
                      ) NOT FINAL NOT INSTANTIABLE
/
CREATE OR REPLACE TYPE Notatip1 UNDER NOTA (
notaTeza number(5,2),
OVERRIDING MEMBER FUNCTION getNotaFin RETURN number) NOT FINAL
```

```
/
CREATE OR REPLACE TYPE Notatip2 under NOTATIP1 (
notaProiect number(5,2),
OVERRIDING MEMBER FUNCTION getNotaFin RETURN number) NOT FINAL
/
CREATE OR REPLACE TYPE BODY Notatip1 as
OVERRIDING MEMBER FUNCTION getNotaFin RETURN number IS
               obj_pnd_deref PonderiNote;
        obj_pnd_real PonderiNote1;
BEGIN
        UTL_REF.SELECT_OBJECT(SELF.ponderi,obj_pnd_deref);
               SELECT  TREAT(obj_pnd_deref AS PonderiNote1) INTO obj_pnd_real  FROM du-
al;
        RETURN  SELF.notaTeza*obj_pnd_real.pndTeza;
EXCEPTION
        WHEN OTHERS  THEN return -1;
END getNotaFin;
end;
/
CREATE OR REPLACE TYPE BODY Notatip2 AS
OVERRIDING MEMBER FUNCTION getnotafin RETURN Number IS
        obj_pnd_deref PonderiNote;
        obj_pnd_real PonderiNote2;
BEGIN
        UTL_REF.SELECT_OBJECT(SELF.ponderi,obj_pnd_deref);
        SELECT TREAT(obj_pnd_deref AS PonderiNote2) INTO obj_pnd_real from dual;
RETURN  SELF.notaTeza*obj_pnd_real.pndTeza+SELF.notaProiect*obj_pnd_real.pndProiect;
EXCEPTION
        WHEN OTHERS THEN  return -1;
END getnotafin;
END;
/
CREATE OR REPLACE TYPE Disciplina AS OBJECT (
        id Integer,
        denumire Varchar2(50),
        modelNota Nota,
        ref_ponderi REF PonderiNote)
/
CREATE TABLE disciplinetbl OF Disciplina (id PRIMARY KEY,
 FOREIGN KEY (ref_ponderi) REFERENCES Ponderitbl);
INSERT INTO disciplinetbl (
 SELECT new Disciplina(1,'Lb. Franceza', new NotaTip1(null,null), REF(p))
               FROM ponderitbl p WHERE p.id=1);
INSERT INTO disciplinetbl (
SELECT  new Disciplina(2,'Informatica Economica',new NotaTip2(null,null,null), REF(p))
FROM ponderitbl p WHERE p.id=2);
INSERT INTO disciplinetbl (
SELECT  new Disciplina(3,'Contabilitate', new NotaTip2(null,null,null), REF(p))
FROM ponderitbl p WHERE p.id=3);
CREATE TABLE Evaluari (Student Varchar2(10),
                       ref_disciplina REF DISCIPLINA REFERENCES Disciplinetbl,
                       dataex DATE, notaex NOTA);
/* note pentru cele trei discipline si doi studenti: */
INSERT INTO  Evaluari (select 'Stud1',ref(d),sysdate, new Notatip1(d.ref_ponderi,7)
             from disciplinetbl d where d.id=1);
INSERT INTO  Evaluari (select 'Stud1',ref(d),sysdate, new Notatip2(d.ref_ponderi,8,10)
             from disciplinetbl d where  d.id=2);
INSERT INTO  Evaluari (select 'Stud1',ref(d),sysdate, new Notatip2(d.ref_ponderi,5,7)
             from disciplinetbl d where  d.id=3);
INSERT INTO  Evaluari (select 'Stud2',ref(d),sysdate,  new Notatip1(d.ref_ponderi,10)
             from disciplinetbl d where d.id=1);
INSERT INTO  Evaluari (select 'Stud2',ref(d),sysdate, new Notatip2(d.ref_ponderi,10,6)
             from disciplinetbl d where  d.id=2);
INSERT INTO  Evaluari (select 'Stud2',ref(d),sysdate, new Notatip2(d.ref_ponderi,8,7)
             from disciplinetbl d where  d.id=3);
```

Se observa în listingul de mai sus ca o tabela obiectuala (`PonderiTbl`) sau un atribut al unei tabele relationale (`Evaluari.notaEx`) pot stoca si obiecte de tipuri derivate din cel declarat, aspect cunoscut si sub numele de substitutie (un obiect de un subtip este stocat de o variabila sau atribut declarate ca fiind de tipul lui superior tocmai pentru a beneficia de generalizare, polimorfism si extensibilitate). Putem spune ca acesta este comportamentul pe care l-am fi asteptat datorita faptului ca un obiect de tip `NotaTip1` nu este decât o alta

speta de `Nota`. Pentru unele prelucrari va trebui totusi sa identificam tipul original al obiectului stocat pentru a avea acces la atributele (sau metodele) lui specializate. Tocmai de aceea în metoda `getNotaFIn()` (listingul 2) apelam la functia `TREAT(ob as Tip)` prin intermediul careia se realizeaza conversia (la nivel declarativ doar) unui obiect de la tipul superior (`PonderiNote`) în tipul original declarat la crearea lui (`PonderiNote1` sau `PonderiNote2`). Functia `TREAT` poate fi utilizata numai în SQL, nefiind deocamdata disponibila în mod direct în PL/SQL.

Concluzii

Începând cu versiunea 9i2, Oracle ofera suport deplin pentru definirea si stocarea obiectelor în baza de date cu tot ceea ce înseamna tehnologia orientata obiect în esenta ei: metode, suprascriere, supraîncarcare, polimorfism. Avantajul major în ceea ce priveste flexibilitatea oferita de aceasta tehnologie aplicatiilor cu baze de date (constrânse pâna nu de mult în schema relationala) nu mai poate fi pus la îndoiala, asa cum a încercat sa demonstreze exemplul din finalul acestui articol.

Bibliografie

Trezzo, C.J. – *Oracle PL/SQL. Tips & Techniques*, McGrawHill, Osborne, 2002
http://otn.oracle.com
http://asktom.oracle.com