PL/SQL – Objektrelationale Konzepte

Objektorientierte vs. objektrelationale DBMS

Motivation zur Verwendung von Objekten in DBS:

- Modellierungen komplexer Aufgabenstellungen mit einfachen Datentypen oft nicht befriedigend
- Abbildung auf komplexe Strukturen und komplexe Methoden erwünscht

Nachteile von ODBMS:

- geringe Verbreitung
- kein einheitlicher Standard für die Schnittstelle zwischen Anwendungsprogramm und Objektdatenbank
- schlechte Performance durch lange Zugriffspfade zu Objekten
  z.B. bei Vererbung und Assoziation
- wenig Unterstützung von Multiuser-Anwendungen

→ Übernahme der Vorteile von objektorientierten Datenbanken in die etablierten relationalen Systeme

→ Kombination beider Paradigmen in einem **objektrelationalen** DBMS

Objektrelationale Datenbanksysteme

Alles funktioniert (fast) genauso wie bisher:

- Spaltenwerte einer Tupeltabelle können Objekte sein
- Einfügen von Objekten:

```
INSERT INTO <objecttable> VALUES (<object-constructor>(attr1, ..., attrn))
```

anstatt

```
INSERT INTO <table> VALUES (attr1, ..., attrn)
```

- Zugriff auf Attribute wie bisher mit `tablename.attr`
- zusätzlich Aufruf von Methoden mit `tablename.meth(...)`

Objektrelationale Konzepte in Oracle

Überblick:

- Definition von Objekttypen
  CREATE TYPE
- Objekttypen und Referenzen
  REF
- Methoden
- Collections
  VARRAY, Nested Table
- Typvererbung
  UNDER
- Polymorphismus
  Overriding, Overloading
- Funktionen und Prädikate für Objekte
  REF, DEREF, TREAT, IS OF

Spezifikation eines Objekttyps

- Schnittstelle des Objekttyps
- Deklaration der Attribute
- Signaturen der Methoden

Syntax:

```
CREATE [OR REPLACE] TYPE <type> AS OBJECT
(
  <attr> <datatype>,
  <attr> <datatype>,
  ...
  MEMBER FUNCTION  <func-name> [(<parameter-list>)] RETURN <datatype>,
  ...
  MEMBER PROCEDURE <proc-name> [(<parameter-list>)],
  ...
);
/
```

Object Types sind Objekte im DB-Schema

Type-Body des Objekttyps

- Implementierung der Objektmethoden
- Signatur muss der in der Spezifikation vorgegebenen entsprechen
- Variable SELF, um auf die Attribute der aktuellen Instanz zuzugreifen

Syntax:

```
CREATE [OR REPLACE] TYPE BODY <type>
AS
   MEMBER FUNCTION <func-name> [( <parameter-list> )] RETURN <datatype>
   IS
   [ <var-decl-list> ;]
   BEGIN
     <PL/SQL-code>
   END;
   . . .

   MEMBER PROCEDURE <proc-name> [( <parameter-list> )]
   IS
   [ <var-decl-list> ;]
   BEGIN
     <PL/SQL-code>
   END;
   . . .
END;
/
```

Beispiel

Spezifikation:

```
CREATE OR REPLACE TYPE person_typ AS OBJECT
( idno  NUMBER,
  name  VARCHAR2(30),
  ort   VARCHAR2(20),
  MEMBER FUNCTION  get_idno RETURN NUMBER,
  MEMBER PROCEDURE display_details
);
```

Type-Body:

```
CREATE TYPE BODY person_typ
AS
    MEMBER FUNCTION get_idno RETURN NUMBER
    IS
    BEGIN
      RETURN idno;
    END;

    MEMBER PROCEDURE display_details
    IS
    BEGIN
        DBMS_OUTPUT.PUT_LINE(TO_CHAR(idno)||' - '||name||' - '||ort);
    END;
END;
```

Verwendung von Objekttypen

- Zeilenobjekte (Objekttabelle):

```
CREATE TABLE person_tab OF person_typ;
```

- Spaltenobjekte (Datentyp für Tabellenspalten):

```
CREATE TABLE department_mgrs
(
  dept_no   NUMBER PRIMARY KEY,
  dept_name CHAR(20),
  dept_mgr  person_typ
);
```

Konstruktor-Methode

- hat gleichen Namen wie der Typ
- genutzt für die Initialisierung

Verwendung:

```
DECLARE
  p1 person_typ;
  p2 person_typ;
BEGIN

  p2 := person_typ(7778,'Diane Smith','Hamburg');
  p1 := person_typ(7777,'John Smith','Leipzig');

END;
/
```

```
CREATE TABLE person_tab OF person_typ;
INSERT INTO person_tab VALUES (person_typ(7777,'John Smith','Leipzig'));
```

Benutzerdefinierter Konstruktor

Beispiel:

```
CREATE OR REPLACE TYPE person_typ AS OBJECT
( idno  NUMBER,
  name  VARCHAR2(30),
  ort   VARCHAR2(20),
  CONSTRUCTOR FUNCTION person_typ(id NUMBER,nm VARCHAR2) RETURN self AS RESULT,
  MEMBER FUNCTION  get_idno RETURN NUMBER,
  MEMBER PROCEDURE display_details
);
```

```
CREATE TYPE BODY person_typ
AS
   CONSTRUCTOR FUNCTION person_typ(id NUMBER,nm VARCHAR2) RETURN self AS RESULT
   IS
   BEGIN
       self.idno := id;
       self.name := nm;
       self.ort  := 'Leipzig';
   RETURN;
   END;

   ...

END;
```

Referenzen - REF DataType

- REF Standard-Datentyp für Attribute: Referenz (logischer Pointer) auf ein ROW-Objekt

```
<ref-attr> REF <object-datatype>
```

- repräsentiert die Objekt-ID des referenzierten Objekts
- Beziehungen zwischen Objekten ohne Foreign Keys
- Naviation durch Objekte mittels Punktnotation (Joins nicht nötig)
  Nutzung: Ändern des referenzierten Objekts
  Lesen einer Kopie eines referenzierten Objekts
  Ändern des REF-Wertes (Verweis auf anderes Objekt oder Null)

REF - Beispiel

Objekttyp:

```
CREATE OR REPLACE TYPE person_typ AS OBJECT
  ( idno    NUMBER,
    name    VARCHAR2(30),
    ort     VARCHAR2(20),
    manager REF person_typ,
    CONSTRUCTOR FUNCTION person_typ(id NUMBER,nm VARCHAR2) RETURN self AS RESULT,
    MEMBER FUNCTION  get_idno RETURN NUMBER,
    MEMBER PROCEDURE display_details
  );
```

Zugriff auf das Referenzobjekt eines Objekts `x` des Typs `person_typ`:

```
  x.manager.name;
```

REF - Beispiel (Fortsetzung)

Tabelle:

```
CREATE TABLE person_tab OF person_typ;
```

Einfügen von Datensätzen **ohne Referenzobjekt**:

```
INSERT INTO person_tab VALUES (person_typ(7777,'John Smith','Leipzig',NULL));
```

Einfügen von Datensätzen **mit Referenzobjekt**:

```
INSERT INTO person_tab values(7778,'Bob Jones','Berlin',(select ref(p) FROM
                              person_tab p WHERE p.idno = 7777));
```

oder:

```
INSERT INTO person_tab
     SELECT person_typ (7779,'Jim Clark','Berlin', REF(p))
     FROM person_tab p
     WHERE p.name = 'John Smith';
```

REF - Beispiel (Fortsetzung)

Nachträgliches Hinzufügen von Referenzen:

```
INSERT INTO person_tab VALUES (person_typ(7780,'Grace Adams','Leipzig',NULL));
```

```
UPDATE person_tab p
SET manager=(SELECT ref(p) FROM person_tab p WHERE p.idno = 7778)
WHERE p.idno=7780;
```

Ausgabe der Datensätze:

```
select p.name, p.manager.name as manager from person_tab p;
```

Vererbung

- Oracle kennt nur einfache Vererbung
- Schlüsselwort UNDER gibt die Oberklasse an, von der abgeleitet wird
- NOT FINAL erlaubt das Anlegen eines Untertypen, FINAL verbietet es
- Object Tables können auch Objekte aller Untertypen speichern
- mit dem Zusatz NOT INSTANTIABLE lassen sich abstrakte Klassen erzeugen, die nur als Oberklasse ohne eigene Instanzen fungieren
- ein Obertyp vererbt alle seine Methoden an die Subtypen
- Im Subtyp: Hinzufügen neuer Attribute
  Hinzufügen neuer Methoden
  Überschreiben/Überladen geerbter Methode

Vererbung - Beispiel

```
CREATE OR REPLACE TYPE person_typ AS OBJECT
( idno  NUMBER,
  name  VARCHAR2(30),
  ort   VARCHAR2(20),
  CONSTRUCTOR FUNCTION person_typ(id NUMBER,nm VARCHAR2) RETURN self AS RESULT,
  MEMBER FUNCTION  get_idno RETURN NUMBER,
  MEMBER PROCEDURE display_details
)NOT FINAL;
```

Subtypen:

```
CREATE TYPE student_typ UNDER person_typ
( matnr NUMBER,
  studg VARCHAR2(30)
) NOT FINAL;
```

```
CREATE TYPE angest_typ UNDER person_typ
( gehalt NUMBER(6,2),
  buero VARCHAR2(30)
) NOT FINAL;
```

Substituierbarkeit

Typen sind substituierbar:
- Supertyp substituierbar durch einen seiner Subtypen
- Subtyp-Instanz ist zugleich auch Instanz des Supertyps

Beispiel:

```
CREATE TABLE person_tab OF person_typ;
```

```
INSERT INTO person_tab VALUES(person_typ(7778,'Bob Jones','Berlin'));

INSERT INTO person_tab VALUES(student_typ(7800,'Joe Lane','Halle',45783,'INB'));

INSERT INTO person_tab VALUES(angest_typ(7784,'Kim Patel','Jena',2478,'Li206'));
```

Die Objekt-Tabelle kann Instanzen aller drei Typen beinhalten.
Der Aufruf der Konstruktor-Methode bestimmt den Typ der jeweils erzeugten Instanz.

Collections - Varray

Varray
- geordnete Menge von Datenelementen des gleichen Datentyps
- Index eines Elements = Positions-Nr. im Array
- beim Anlegen wird Initialgröße spezifiziert, kann bei Bedarf vergrößert werden
- Physikalische Speicherung erfolgt z.B. als BLOB

Beispiel:

```
CREATE TYPE email_list_arr AS VARRAY(10) OF VARCHAR2(80);
```

Verwendung:
- Spaltentyp einer Tabelle (damit keine 1NF mehr)
- Attribut eines Objekttyps
- Typ einer PL/SQL Variablen,
- Parameter oder Rückgabewert einer Funktion

Collections - Nested Table

Nested Table

- Dynamische Datenstruktur beliebiger Größe
- Ungeordnete Menge von Datenelementen des gleichen Typs
- Hat nur eine Spalte (Basistyp oder benutzerdefiniert)
- Zugriff erfolgt über SELECT, INSERT, DELETE und UPDATE
- physikalische Speicherung erfolgt in *Storage Tables* (Datenbanktabellen)

Beispiel:

```
CREATE TYPE person_table_typ AS TABLE OF person_typ;
```

```
CREATE TABLE abteilungen
(
    abtno NUMBER,
    abtname VARCHAR2(20),
    team  person_table_typ
)
NESTED TABLE team STORE AS team_tab;
```

Collections - Nested Table (Forts.)

Einfügen von DS:

```
INSERT INTO abteilungen VALUES
(100,'Development',
 person_table_typ( person_typ(7777,'John Smith','Leipzig'))
);
```

Anfragen:
Ausgabe aller Abteilungen und Team-Mitglieder

```
SELECT a.abtname, t.*
FROM abteilungen a, table(a.team) t
ORDER BY a.abtname, t.name;
```

Ändern von Daten:

```
UPDATE TABLE ( SELECT a.team
               FROM abteilungen a
               WHERE a.abtno=100) t
SET t.name= 'Diane Miller'
WHERE t.idno=7776;
```

Ausgabefunktionen

- **VALUE**
  In: Tabellen-Alias für Object Table
  Out: Objektinstanzen, die in Object Table als Rows enthalten sind

```
SELECT VALUE(p)
FROM person_tab p
WHERE p.name = 'Bob Jones';
```

- **REF**
  In: Tabellen-Alias für Object Table
  Out: Referenz zu einer Instanz

```
SELECT REF(p)
FROM person_tab p
WHERE p.idno = 7780 ;
```

- **DEREF**
  liefert die Objekt-Instanz, auf die eine REF verweist

```
SELECT DEREF(REF(p))
FROM person_tab p;
```

Ausgabefunktionen (Forts.)

TREAT

- zur Typanpassung (meist: Subtyp statt Supertyp)
  z.B. Behandlung einer Person als Student

Zwei Anwendungen:

- bei Zuweisungen an Variablen spezialisierter Typen
  (Supertyp-Wert → Subtyp)
- Zugriff auf Attribute oder Methoden des Subtyps

Beispiel:

```
SELECT name, TREAT(VALUE(p) AS student_typ).studg studiengang
FROM person_tab p;
```

```
NAME           STUDIENGANG
-----------    ----------------
Bob Jones      null
Joe Lane       INB
Kim Patel      null
```

Ausgabefunktionen (Forts.)

IS OF type

- Prädikat zum Test, ob Objekt-Instanz zum jeweiligen Subtyp gehört

```
SELECT value(p)
FROM person_tab p
WHERE VALUE(p) IS OF (student_typ);
```

Ausgabe:

```
VALUE(p)
----------------------------------------------
STUDENT_TYP(7800,'Joe Lane', ...                )
```