Tipps & Tricks: Objektorientierung

Bereich:	PL/SQL	Erstellung:	03/2001 HA
Versionsinfo:	9.2, 10.2, 11.1	Letzte Überarbeitung:	06/2009 HA

Objektorientierung unter Oracle

Objektorientierung unter Oracle richtet sich einerseits nach objekt-orientierten Programmiersprachen wie Java, andererseits nach dem Package-Konzept aus PL/SQL.

Voraussetzung für die Erstellung von Instanzen (= Objekten) ist die Schaffung von Objekttypen innerhalb der Datenbank. Diese Typen beinhalten Attribute und evtl. Methoden. Eine Methode kann entweder eine Prozedur oder eine Funktion sein, wobei der dazugehörige Quellcode analog zum Package-Konzept erst im Typ-Body erscheint. Man unterscheidet zwischen STATIC und MEMBER Methoden. Analog zu Java werden STATIC Methoden unabhängig von einer Instanz aufgerufen, während MEMBER Methoden nur im Zusammenhang mit einer Instanz aufgerufen werden können und den impliziten Parameter "self" kennen. Der Aufruf erfolgt über Punktnotation.

Ein Body ist nur dann nötig, wenn die Spezifikation auch Methoden (ohne call_spec) enthält. Eine Methode kann auch aus einem Aufruf einer C- oder (ab Version 8i) Java-Routine bestehen (= call_spec).

Eine Instanz wird erzeugt durch Aufruf des Konstruktors, an den Werte für alle (bzw. alle erforderlichen) Attribute in der richtigen Reihenfolge übergeben werden müssen. Konstruktorname ist dabei gleich Typname. Es gibt immer zumindest einen implizit deklarierten Konstruktor; an diesen müssen für ALLE Attribute Werte übergeben werden; seit Version 9.2 gibt es auch benutzerdefinierte Konstruktoren.

Übersicht der Themen:

Anlegen eines Objekttyps

Syntax (unvollständig):

```
CREATE [OR REPLACE] TYPE type_name
   [AUTHID {CURRENT_USER | DEFINER}] {IS | AS} OBJECT (
   attribute_name datatype[, attribute_name datatype]...
   [{MAP | ORDER} MEMBER function_spec,]
   [{MEMBER | STATIC} {subprogram_spec | call_spec}
   [, {MEMBER | STATIC} {subprogram_spec | call_spec}]...]
);
/
[CREATE [OR REPLACE] TYPE BODY type_name {IS | AS}
   { {MAP | ORDER} MEMBER function_body;
      | {MEMBER | STATIC} {subprogram_body | call_spec};}
```

```
   [{MEMBER | STATIC} {subprogram_body | call_spec};]...
END;
/
]
```

Soll ein Typ Vererbung zulassen, so muss am Ende ergänzt werden: NOT FINAL

Leitet sich ein Typ von einem Supertyp ab, so wird statt {IS | AS} OBJECT angegeben UNDER <Supertyp>.

Beispiel:

```
CREATE TYPE Person AS OBJECT (
   vorname    VARCHAR2(20),
   nachname   VARCHAR2(30),
   l_alter     NUMBER,
   geschlecht VARCHAR2(1),
   strasse    VARCHAR2(30),
   plz        NUMBER,
   ort        VARCHAR2(30),
   MEMBER PROCEDURE neuerNachname(neuerName VARCHAR2),
   STATIC FUNCTION geburtsjahr(p_alter NUMBER) RETURN NUMBER);
/
CREATE TYPE BODY Person AS
   MEMBER PROCEDURE neuerNachname(neuerName VARCHAR2) IS
   BEGIN
      self.nachname := neuerName;
   END;
   STATIC FUNCTION geburtsjahr(p_alter NUMBER) RETURN NUMBER IS
      jahr NUMBER;
   BEGIN
      jahr := TO_NUMBER(TO_CHAR(SYSDATE, 'RRRR'));
      RETURN (jahr - p_alter);
   END;
END;
/
```

Benutzerdefinierte Konstruktoren

Bis einschließlich Version 9i konnte eine Instanz eines Objekts nur über den Aufruf des implizit erzeugten Konstruktors geschaffen werden. Seit Version 9i Release 2 ist es möglich, zusätzlich eigene Konstruktoren zu schreiben. Das hat den Vorteil, dass nicht immer alle Attribute mitgegeben werden müssen beim Aufruf, sondern Standardwerte vorgegeben werden können.

Beispiel:

```
CREATE OR REPLACE type Person AS OBJECT (
   vorname      VARCHAR2(20),
   nachname     VARCHAR2(30),
   l_alter      NUMBER,
   geschlecht   VARCHAR2(1),
   strasse      VARCHAR2(30),
```

```
   plz          NUMBER,
   ort          VARCHAR2(30),
   MEMBER PROCEDURE neuerNachname(neuerName VARCHAR2),
   STATIC FUNCTION geburtsjahr(p_alter NUMBER) RETURN NUMBER,
   CONSTRUCTOR FUNCTION Person(vorname      VARCHAR2,
                               nachname     VARCHAR2,
                               l_alter      NUMBER,
                               geschlecht   VARCHAR2,
                               strasse      VARCHAR2)
   RETURN SELF AS RESULT,
   CONSTRUCTOR FUNCTION Person(vorname      VARCHAR2,
                               nachname     VARCHAR2,
                               l_alter      NUMBER,
                               geschlecht   VARCHAR2)
   RETURN SELF AS RESULT);
/
CREATE OR REPLACE TYPE BODY Person AS
   MEMBER PROCEDURE neuerNachname(neuerName VARCHAR2) IS
   BEGIN
      self.nachname := neuerName;
   END;

   STATIC FUNCTION geburtsjahr(p_alter NUMBER) RETURN NUMBER IS
      jahr NUMBER;
   BEGIN
      jahr := TO_NUMBER(TO_CHAR(SYSDATE, 'RRRR'));
      RETURN (jahr - p_alter);
   END;

   CONSTRUCTOR FUNCTION Person(vorname      VARCHAR2,
                               nachname     VARCHAR2,
                               l_alter      NUMBER,
                               geschlecht   VARCHAR2,
                               strasse      VARCHAR2)
                               RETURN SELF AS RESULT IS
   BEGIN
      SELF.vorname:= vorname;
      SELF.nachname := nachname;
      SELF.l_alter := l_alter;
      SELF.geschlecht := geschlecht;
      SELF.strasse := strasse;
      SELF.plz := 82008;
      SELF.ort := 'Unterhaching';
      RETURN;
   END;
   CONSTRUCTOR FUNCTION Person(vorname      VARCHAR2,
                               nachname     VARCHAR2,
                               l_alter      NUMBER,
                               geschlecht   VARCHAR2)
                               RETURN SELF AS RESULT IS
   BEGIN
      SELF.vorname:= vorname;
      SELF.nachname := nachname;
      SELF.l_alter := l_alter;
```

```
      SELF.geschlecht := geschlecht;
      SELF.strasse := 'Grünwalder Weg';
      SELF.plz := 82008;
      SELF.ort := 'Unterhaching';
      RETURN;
   END;
END;
/
```

Ändern eines Objekttyps

Der Body kann jederzeit mit CREATE OR REPLACE an die Spezifikation angepasst werden. Im CREATE TYPE-Befehl der Spezifikation dagegen ist die Option OR REPLACE nur so lange zulässig, wie keine abhängigen Objekte (z.B. Tabellen mit einer Spalte dieses Datentyps) existieren. Wenn bereits Abhängigkeiten bestehen, sind Änderungen in der Spezifikation nur noch mit dem ALTER-Befehl möglich.

Mit dem ALTER-Befehl können gezielt einzelne Attribute oder Methoden ergänzt (ADD) oder gelöscht (DROP) werden. Attribute können auch geändert (MODIFY) werden, z. B. kann aus einem VARCHAR2(20)-Attribut nachträglich ein VARCHAR2(50)-Attribut gemacht werden.

Alternativ kann ein Objekttyp mit ALTER ..REPLACE geändert werden. Dabei gelten folgende Einschränkungen:

- Es können weder Attribute noch Methoden entfernt werden
- Es können nur Methoden ergänzt werden, keine Attribute
- Attribute und Methoden müssen vollständig wieder aufgelistet werden
- Es darf noch kein Attribut verändert worden sein

In alten Versionen (bis einschließlich 8.1.7) war REPLACE die einzige Möglichkeit, einen Objekttyp (mit abhängigen Objekten) nachträglich zu ändern.

Beispiele:

```
ALTER TYPE Person REPLACE AS OBJECT (
   vorname    VARCHAR2(20),
   nachname   VARCHAR2(30),
   l_alter    NUMBER,
   geschlecht VARCHAR2(1),
   strasse    VARCHAR2(30),
   plz        NUMBER,
   ort        VARCHAR2(30),
   MEMBER PROCEDURE neuerNachname(neuerName VARCHAR2),
   STATIC FUNCTION geburtsjahr(p_alter NUMBER) RETURN NUMBER,
   MEMBER PROCEDURE Umzug(neueStrasse VARCHAR2,
                          neuePLZ     NUMBER,
                             neuerOrt     VARCHAR2));

CREATE OR REPLACE TYPE BODY Person AS
   MEMBER PROCEDURE neuerNachname(neuerName VARCHAR2) IS
   BEGIN
      self.nachname := neuerName;
   END;
   STATIC FUNCTION geburtsjahr(p_alter NUMBER) RETURN
```

```
   NUMBER IS
      jahr NUMBER;
   BEGIN
      jahr := TO_NUMBER(TO_CHAR(SYSDATE, 'RRRR'));
   RETURN (jahr - p_alter);
   END;
   MEMBER PROCEDURE Umzug(neueStrasse VARCHAR2,
                    neuePLZ    NUMBER,
                    neuerOrt    VARCHAR2) IS
   BEGIN
      self.strasse := neueStrasse;
      self.PLZ := neuePLZ;
      self.ort := neuerOrt;
   END;
END;
/

ALTER TYPE Person ADD MEMBER PROCEDURE Umzug
        (neueStrasse VARCHAR2,
         neuePLZ     NUMBER,
         neuerOrt     VARCHAR2);
ALTER TYPE Person DROP MEMBER PROCEDURE Umzug
          (neueStrasse VARCHAR2,
         neuePLZ     NUMBER,
          neuerOrt     VARCHAR2);
ALTER TYPE Person ADD ATTRIBUTE (land VARCHAR2(20));
ALTER TYPE Person MODIFY ATTRIBUTE (land VARCHAR2(50));
ALTER TYPE Person DROP ATTRIBUTE (land);
```

Auch eine Kombination der verschiedenen Klauseln und die Angabe mehrerer Attribute bzw. Methoden sind möglich. Falls bereits abhängige Objekte existieren, muss zusätzlich angegeben werden, wie mit diesen zu verfahren ist. Dies geschieht mit einer Klausel am Ende des ALTER TYPE-Befehls:

- `INVALIDATE` setzte alle abhängigen Objekte auf INVALID.
- `CASCADE` gibt die Änderungen weiter. Dazu müssen allerdings alle abhängigen Objekte rekompiliert werden. Tritt dabei ein Problem auf, kann die Änderung nicht durchgeführt werden. Durch zusätzliche Angabe von FORCE (plus Exceptions-Tabelle) können die Änderungen jedoch erzwungen werden.
- `CASCADE NOT INCLUDING TABLE DATA` gibt die Änderungen weiter, doch werden abhängige Tabellen zunächst auf INVALID gesetzt und nicht angepasst. Sie müssenspäter mit `ALTER TABLE <tabellenname> UPGRADE INCLUDING DATA;` konvertiert werden.Eine Änderung der Methoden ist unproblematisch.

Tipps

- Jeder ALTER TYPE-Befehl, der ein Attribut betrifft, führt zu einem zusätzlichen Eintrag in DBA_OBJECTS
- Jeder ALTER TYPE-Befehl (ausser REPLACE) findet sich explizit in DBA_SOURCE wieder

Vererbung

Vererbung heisst, ein abgeleiteter Objekttyp (ein sogenannter Subtyp) beinhaltet automatisch alle Attribute und Methoden des Supertyps, von dem er abgeleitet wurde. Oracle unterstützt ausschließlich Vererbung in einer Linie: ein Subtyp kann nur von einem Supertypen abgeleitet werden, nicht von mehreren. Andererseits können

sich beliebig viele Subtypen vom gleichen Supertyp ableiten. Zusätzlich zu den ererbten Attributen können neue Attribute ergänzt werden, ererbte Methoden können überschrieben und/oder durch zusätzliche Methoden ergänzt werden.

Deklaration von Supertypen

Damit ein Objekttyp vererben (d.h., als Supertyp dienen) kann, muss er explizit als NOT FINAL deklariert werden, der Default ist FINAL. Auch einzelne Methoden innerhalb der Typdeklaration können als FINAL bzw. NOT FINAL deklariert werden. Eine Methode allerdings ist per Default NOT FINAL. Wird sie explizit als FINAL deklariert, kann sie von keinem Subtyp überschrieben werden.

Beispiel:

```
CREATE TYPE Person AS OBJECT (
   vorname         VARCHAR2(20),
   nachname        VARCHAR2(30),
   l_alter         NUMBER,
   geschlecht      VARCHAR2(1),
   strasse         VARCHAR2(30),
   plz             NUMBER,
   ort             VARCHAR2(30),
   MEMBER PROCEDURE neuerNachname(neuerName VARCHAR2),
   FINAL STATIC FUNCTION geburtsjahr(p_alter NUMBER) RETURN NUMBER)
   NOT FINAL;
/
```

oder nachträglich:

```
ALTER TYPE Person NOT FINAL [CASCADE|INVALIDATE];
```

Deklaration von Subtypen

Ein Subtyp wird angelegt mit der Klausel UNDER <supertyp>. Weitere Attribute und Methoden können zusätzlich angegeben werden. Um Subtypen zu jedem beliebigen Supertyp anlegen zu können, benötigt ein User das Privileg UNDER ANY TYPE.

Overloading und Overriding

Subtypen können Methoden gleichen Namens nur dann zusätzlich deklarieren, wenn sich ihre Signatur von der ererbten Methode unterscheidet. Hier gelten die gleichen Regeln wie beim Overloading innerhalb von Packages.

Subtypen können eine ererbte Prozedur auch überschreiben ("Overriding"), soweit diese nicht explizit als FINAL deklariert wurde. Eine STATIC-Methode darf dabei allerdings nicht durch eine MEMBER-Methode überschrieben werden, und umgekehrt.

Beispiel:

```
CREATE TYPE Mitarbeiter UNDER Person
   (ma_nr      NUMBER,
   abteilung    VARCHAR2(30),
   vorgesetzter NUMBER,
   OVERRIDING MEMBER PROCEDURE neuerNachname(neuerName VARCHAR2)
   )
```

```
  NOT FINAL;
/
```

In diesem Beispiel müsste im zugehörigen Body die Prozedur neuerNachname neu implementiert werden. Anmerkung: Würde in diesem Beispiel "vorgesetzter" als "Person" definiert werden (was durchaus zulässig ist), könnte in einer relationalen Tabelle keine Spalte mehr vom Datentyp "Person" sein.

NOT INSTANTIABLE

Sowohl eine Methode als auch ein Typ können als NOT INSTANTIABLE deklariert werden. Wird eine Methode als NOT INSTANTIABLE deklariert, muss auch der Typ zwingend als NOT INSTANTIABLE deklariert werden.

Ein Typ, der als NOT INSTANTIABLE deklariert wurde, verfügt über keinen Konstruktor, es können also keine Instanzen von diesem Typ angelegt werden.
Ein solcher Typ muss zwingend als NOT FINAL deklariert werden; sie machen auch nur Sinn als Supertypen.

Eine Methode, die als NOT INSTANTIABLE deklariert wurde, ist nur deklariert, aber nicht implementiert. Es bleibt den Subtypen überlassen, diese Methode zu implementieren. In diesem Fall muss die Definition mit OVERRIDING erfolgen. Wenn nicht, muss auch der Subtyp als NOT INSTANTIABLE deklariert werden.

Beispiel:

```
CREATE TYPE keine_instanz AS OBJECT (id NUMBER,
       MEMBER PROCEDURE imp,
       NOT INSTANTIABLE MEMBER PROCEDURE nicht_imp)
       NOT INSTANTIABLE NOT FINAL;
/
```

Hinweis:

- NOT INSTANTIABLE Typen sind mit Interfaces in Java vergleichbar.
- Auch [NOT] INSTANTIABLE kann nachträglich geändert werden mit `ALTER TYPE <typname> [NOT] INSTANTIABLE [INVALIDATE|CASCADE]`;

Ersetzbarkeit

Ein Objekt eines Subtyps kann an Stelle eines Objekts eines übergeordneten Supertyps eingesetzt werden, aber nicht umgekehrt. Dies gilt für Spalten in relationalen Tabellen, Zeilen in Objekttabellen, Views und auch für REF-Spalten auf entsprechende Objekte.

Beispiele zu Objekten in PL/SQL

Die Beispiele gehen von obiger Deklaration des Objekttyps PERSON aus. Im ersten Beispiel wird der implizite Konstruktor verwendet, im zweiten werden benutzerdefinierte Kontruktoren aufgerufen.

Beispiel 1:

```
SET SERVEROUTPUT ON
DECLARE
```

```
   p     Person;
   jahrgang NUMBER;
BEGIN
   -- Aufruf des impliziten Konstruktors:
   p := Person ('Verena', 'Meier', 38, 'w', 'Hauptstraße 12', 4711,
                 'echt kölnisch');
   DBMS_OUTPUT.PUT_LINE('Nachname von p ist '||p.nachname);
   -- Aufruf einer MEMBER Methode:
   p.neuerNachname('Neubusch');
   DBMS_OUTPUT.PUT_LINE('neuer Nachname von p ist '||p.nachname);
   -- Aufruf einer STATIC Methode
   jahrgang := Person.geburtsjahr(47);
   DBMS_OUTPUT.PUT_LINE('Eine 47-jährige Person wurde '||jahrgang||
                         ' geboren');
 END;
/
```

Beispiel 2 :

```
SET SERVEROUTPUT ON
DECLARE
    p person;
    p2 person;
BEGIN
   p := person('Hugo', 'Meier', 23, 'm','Hauptstrasse');
   DBMS_OUTPUT.PUT_LINE('Wohnort: '|| p.ort);
   p2 := person('Egon', 'Müller', 23, 'm');
   DBMS_OUTPUT.PUT_LINE('Strasse: '|| p2.strasse);
END;
/
```

Beispiele zu Objekten in Tabellen

Auch diese Beispiele gehen von obigen Deklarationen aus. Verwendet werden die Objekttypen PERSON und MITARBEITER.

Relationale Tabellen

Instanzen von Objekttypen sind nicht nur in PL/SQL zulässig. Auch Spalten einer Tabelle können als Datentyp einen Objekttyp haben:

anlegen:

```
CREATE TABLE kunden (knd_nr NUMBER,
                 kunde  Person);
```

abfragen:

```
SELECT * FROM kunden;
```

```
SELECT k.knd_nr, k.kunde.nachname FROM kunden k;
```

DML:

```
--CREATE SEQUENCE knd_seq;
INSERT INTO kunden
VALUES (knd_seq.NEXTVAL,
   Person('Verena',
         'Meier',
         38,
         'w',
         'Hauptstraße 12',
         4711,
         'echt kölnisch'));

UPDATE kunden k
   SET k.kunde.l_alter = 39
 WHERE k.knd_nr = 1;

DELETE FROM kunden k
 WHERE k.knd_nr = 1;

DELETE FROM kunden k
 WHERE k.kunde.nachname = 'Meier';
```

Um hier unmittelbar auf ein einzelnes Attribut eines Objekts zugreifen zu können, ist ein Tabellen-Alias zwingend erforderlich.

Beispiel für Ersetzbarkeit:

```
--CREATE SEQUENCE ma_nr;
INSERT INTO kunden
VALUES (knd_seq.NEXTVAL,
       Mitarbeiter('Verena',
               'Meier',
               38,
               'w',
               'Hauptstraße 12',
                    4711,
               'echt kölnisch',
                 ma_nr.NEXTVAL,
               'Vertrieb',
                127));
```

Object Tables

Daneben gibt es auch object tables, die pro Zeile ein Objekt des vorgegebenen Objekttyps enthalten. Eine Spalte entspricht dabei einem Attribut des Objekts:

anlegen:

```
CREATE TABLE kollegen OF Person;
```

abfragen:

```
SELECT * FROM kollegen;
SELECT nachname FROM kollegen;
```

Um nicht nur die Attribute, sondern das gesamte Objekt auszulesen, ist die Funktion VALUE nötig:

```
SELECT VALUE(k) [INTO var] FROM kollegen k;
-- wobei die Variable var vom Datentyp Person sein muss
```

DML:

```
INSERT INTO kollegen
VALUES (Person('Verena',
            'Meier',
            38,
            'w',
            'Hauptstraße 12',
                 4711,
            'echt kölnisch'));
```

Für die object table ist auch die vereinfachte Form zulässig:

```
INSERT INTO kollegen
VALUES ('Verena',
       'Meier',
       38,
      'w',
      'Hauptstraße 12',
       4711,
        'echt kölnisch');

UPDATE kollegen
   SET nachname = 'Huber'
 WHERE nachname = 'Meier';

DELETE FROM kollegen
 WHERE nachname = 'Meier';
```