Objekte machen das Leben leichter

Jürgen Sieben

In dieser Folge möchte der Autor eine Lanze für die recht selten genutzte Objektorientierung in PL/SQL brechen.

Ein erstes Beispiel wurde bereits verwendet, um mithilfe einer „NESTED TABLE“ beliebig viele Parameter an eine „BULK REPLACE“-Prozedur zu übergeben (siehe Red Stack Magazin, Ausgabe 06/2017). Schon dort war die Tatsache angenehm aufgefallen, dass Objekte gleichermaßen in SQL und PL/SQL verwendet werden können. Diesen Ansatz soll ein wenig weitergedacht werden.

Oracle unterstützt drei Typen von Objekten: das eigentliche „OBJECT“, das einem Record ähnelt, und die beiden Kollektionstypen „VARRAY“ (variabel langes Array, dem eine maximale Anzahl von Elementen mitgegeben wird und die eine garantierte Sortierung hat) und die unbegrenzte, aber auch nicht sicher sortierte „NESTED TABLE“.

Ein „OBJECT“ kann immer auch als Ersatz für einen Record verwendet werden, wenn man ausschließlich Attribute im Objekt festlegt und einen Standard-Konstruktor verwendet, um ein Objekt zu erstellen: Ähnlich wie die Funktion „TO_DATE“ aus einer Zeichenkette eine Instanz vom Typ „DATE“ erstellt, erzeugt ein Konstruktor eine Instanz des Objekts. Der Standard-Konstruktor heißt genauso wie der Typ, den er erstellen soll, und erwartet, dass alle Attribute des Objekts mit Werten belegt werden. Hiervon kann man jedoch auch abweichen und eigene Konstruktor-Methoden erzeugen, etwa um nur einige Parameter zu übergeben und die Attributwerte aus diesen Parametern zu ermitteln.

Ein Beispiel: Ein Logging-Package soll in der Lage sein, Meldungen durch eine beliebige Anzahl von Parametern zu spezialisieren; zudem soll die Meldung in der Sprach-Einstellung der Session erzeugt werden. Dies kann natürlich auch über herkömmlichen Code erfolgen, alternativ wäre es allerdings auch möglich, einem Objekt vom Typ „MESSAGE_TYPE“ dieses Verhalten beizubringen. *Listing 1* zeigt ein solches Objekt, man erkennt die Konstruktor-Methode, die für die Instanziierung dieses Objekts verwendet werden soll.

Beim genauen Betrachten ist ein Attribut „P_ARG_LIST“ vom Typ „MSG_ARGS“ zu erkennen. „MSG_ARGS“ ist ein Kollektionstyp, allerdings keine „NESTED TABLE“, sondern ein variables Array („VARRAY“) mit bis zu zwanzig Einträgen. In einem Objekt können also weitere Objekt-Typen als Attribut hinterlegt sein. Die Konstruktor-Methode übernimmt nun die Aufbereitung der Nachricht in der benötigten Sprache und liefert eine Instanz der Meldung zurück, die nicht nur aus dem Meldungstext, sondern auch aus flankierenden Informationen besteht, die in der weiteren Verarbeitung benötigt werden. Umgesetzt sind diese Konstruktor-Methoden als Methoden mit der Klausel „CONSTRUCTOR FUNCTION“, die „SELF“ als Ein- und Ausgabe-Parameter besitzen. Das ist die Objekt-Instanz, die über die Methode mit der Anweisung „RETURN“ später zurückgegeben werden wird.

Bevor wir den Code zur Erzeugung eines Objekts betrachten, zunächst die Fragen: „Wozu das alles?“, „Warum erstellen wir eine Meldung nicht mit normalem Code?“. Der wichtigste Aspekt ist die Kapselung der Logik. Wie eine Meldung konkret gebaut wird, weiß der Datentyp selbst. Sobald das Objekt kompiliert ist, lässt es sich wie ein eingebauter Datentyp verwenden, die zugehörige Logik gibt es unmittelbar dazu. Alternativ wäre die Erstellung einer Meldung nur über ein Package möglich, doch dann hätte man noch einen Plan entwickeln müssen, wie die weiteren Attribute bei der Meldung gespeichert sind. Diese Alternativlösung hätte einen Record deklariert und diesen in einem Hilfspackage mit Werten gefüllt. Das kann ein Objekt

```
create or replace type message_type force is object(
   id number,
   message_name varchar2(30 char),
   message_text clob,
   severity number(2,0),
   stack varchar2(2000),
   backtrace varchar2(2000),
   error_number number (5,0),
   message_args msg_args,
   -- Konstruktor
   constructor function message_type(
       self in out nocopy message_type,
       p_message_name in varchar2,
       p_arg_list msg_args)
   return self as result)
;
```

Listing 1

```
create or replace type body message_type
as
   constructor function message_type(
       self in out nocopy message_type,
       p_message_name in varchar2,
       p_arg_list msg_args)
       return self as result
   as
       l_errm varchar2(2000);
   begin
       -- Meldungstext aus Meldungstabelle lesen und Sprache festlegen
       select pms_pml_name,
              pms_text,
              pms_pse_id,
              coalesce(pms_active_error, pms_custom_error) error_number
           from pit_message m
           join pit_message_language_v l -- View liefert die Sessionsprache
               on m.pms_pml_name = l.pml_name
           where m.pms_name = p_message_name
           order by l.pml_default_order desc
           fetch first 1 row only;

       -- Objektattribute belegen
       self.id := pit_log_seq.nextval;
       self.message_name := p_message_name;
       self.message_args := p_arg_list;

       -- Im Fehlerfall: Fehlerdetails speichern
       if sqlcode > 0 and self.severity <= 30 then
           self.stack := dbms_utility.format_error_stack;
           self.backtrace := dbms_utility.format_error_backtrace;
       end if;

       -- Platzhalter der Meldung durch Variablen ersetzen (bulk replace)
   if p_arg_list is not null then
       for i in p_arg_list.first..p_arg_list.last loop
           self.message_text :=
               replace(self.message_text, '#' || i || '#', p_arg_list(i));
           end loop;
       end if;
       return;
   end;
end;
Nun kann das Objekt sehr einfach verwendet werden:
declare
   l_msg message_type;
begin
   l_msg := message_type('MY_MESSAGE', msg_args('FOO', 'ACME'));
end;
```

Listing 2

direkt und vor allem: auch in SQL. Selbst wenn man einen Record erstellt hätte, wäre dieser nicht in SQL verwendbar und könnte auch nicht in einer Tabelle gespeichert werden. Dies wäre für Meldungen aber durchaus praktisch, zum Beispiel weil nun ein Log-Eintrag auch im Nachhinein in unterschiedlichen Sprachen aufbereitet werden kann, und das geht eben nur mit Objekten.

Listing 2 zeigt die Implementierung des Objekt-Typs, die ähnlich einem Package im „TYPE BODY“ erfolgt. Über den Parameter „SELF“ werden die Objekt-Attribute belegt, und dieses Objekt liefert zum Ende der Methode über die Anweisung „RETURN“ zurück. In unserem Fall wird die Meldung aus einer Meldungs-Tabelle ermittelt und die Spracheinstellung dynamisch aus einer Voreinstellung und der Session-Sprache abgeleitet. Bei aufmerksamem Studium des Codes ist die „BULK REPLACE“-Methode wiederzuerkennen, die hier alle übergebenen Parameter im Meldungstext auf Meldungsanker verteilt.

Fazit

Objekte können ein Ersatz für Records sein, haben diesen aber einen Standard-Konstruktor voraus, der das Belegen der Attribute einfacher macht als bei einem Record, zudem können Objekte in SQL und PL/SQL genutzt werden. Wird mehr Logik benötigt, um eine Instanz des Objekts zu erstellen, können eigene (und auch beliebig viele) Konstruktor-Methoden erzeugt werden, um die für die Erstellung benötigte Logik beim Objekt zu halten und dort zu kapseln. Ein Package ist im Regelfall nicht erforderlich.

Die Möglichkeit, Logik in Objekten zu abstrahieren, ist erfahrungsgemäß für viele PL/SQL-Entwickler eine ungewohnte Lösung, doch der Autor hofft, durch dieses einfache Beispiel einige der Vorteile dieser Lösung aufgezeigt zu haben. Es lohnt sich, sich an Objekte heranzutrauen.

Jürgen Sieben
j.sieben@condes.de