# The Brave New World of PL/SQL 9i

**The Brave New World of PL/SQL 9i' was de titel van de presentatie die de Amerikaan Steven Feuerstein op 11 april jl. heeft gegeven voor de leden van de OGH. In 1999 en 2000 was hij eveneens te gast om zijn uitgebreide kennis van Oracle PL/SQL met de OGh-leden te delen. Feuerstein wordt gezien als één van 's werelds toonaangevende experts op het gebied van Oracle's procedurele taal en staat bekend als een onderhoudend spreker. Niet voor niets zat de Morse zaal van Oracle in De Meern dan ook helemaal vol. In dit artikel worden de nieuwe PL/SQL elementen van Oracle 9i nog eens vermeld, met hier en daar een verdieping.**

Feuerstein vulde de presentatie met enkele van de volgende onderwerpen:

- Geïntegreerde SQL en PL/SQL parsers
- Ondersteuning voor Object Type inheritance (overerving)
- Nieuwe datatypes en built-in functies
- Table functions en cursor expressions
- De CASE, NVL2 en MERGE statements
- XML ondersteuning
- PL/SQL Native compilation

Tijdens de presentatie bleek dat een groot deel van de aanwezigen nog maar nauwelijks echt op de hoogte was van de Oracle 8i PL/SQL uitbreidingen, zodat veel van de vragen na afloop leidden tot een korte uitleg van enkele van de meest belangrijke 8i elementen, zoals Autonomous Transactions, Virtual Private Database en de BULK en FORALL statements. Oracle stoomt blijkbaar hard door met het verbeteren/uitbreiden van zijn bekendste product, het RDBMS en niet iedereen is in staat dit tempo bij te houden. Het wel of niet in de praktijk kunnen werken aan reële opdrachten met de nieuwste versies zal hierin uiteraard een belangrijke factor zijn.

## Geïntegreerde SQL en PL/SQL parsers

In 9i zijn de parsers van SQL en PL/SQL geïntegreerd, wat wil zeggen dat alles wat direct in SQL mogelijk is, ook in 'embedded' SQL (SQL binnen PL/SQL modules) te gebruiken is. Wie eens geprobeerd heeft het in Oracle 8i geïntroduceerde CASE statement - dat prima werkt in SQL - te gebruiken in embedded SQL, heeft gezien dat dit niet werkt. In Oracle 9i derhalve wel.

De geïntegreerde parsers hebben mogelijk wel tot gevolg dat stukken PL/SQL die in voorgaande versies werkten, in 9i niet meer compileren. Het betreft hier wel constructies die niet direct voldoen aan 'Best practices', maar die Oracle toestond vóór 9i, maar in 9i niet meer. Een instance parameter geeft de mogelijkheid om dit in 9i 'alsnog' toe te staan zolang de code nog niet is aangepast.

## Object Option

Eén van de PL/SQL onderdelen die, zeker in Europa, nog heel weinig aandacht hebben gehad in de praktijk en waarvan Feuerstein al aangaf dat dit wellicht komt omdat het in eerdere versies nog te onvolledig uitgewerkt was, is de Object Option. In Versie 8.0 van zijn RDBMS introduceerde Oracle Object Types (Classes in veel andere OO-talen), maar het duurde tot versie 9 voordat één van de meest in het oog springende kenmerken van OO ondersteund werd, namelijk overerving (Engels: inheritance). Een Object Type in Oracle is een soort template voor instanties van dat Type. Een Object Type definieert zowel de attributen (data) als de members (methodes, in Oracle termen 'procedures' en 'functies') en alle instanties van het Object Type bevatten dan ook deze attributen en van alle instanties kunnen members worden 'aangeroepen'.

De toevoeging van overerving maakt de Object Option in één klap een stuk interessanter. Via overerving is het mogelijk een hiërarchie van Object Types te maken, ofwel Sub types van andere Object Types. Een Subtype is een specifieker type dan zijn Parent Type. Een simpel voorbeeld is bijvoorbeeld een 'persoon', wat een algemeen Type is en waartoe wij allemaal behoren, en 'leraar', wat ook een persoon is, maar een specifieker type persoon, namelijk iemand die les geeft. Degenen aan wie de leraar les geeft zijn van het type 'leerling'. 'Leraar' en 'leerling' kunnen dan gemodelleerd worden als subtypes van Type 'persoon'.
Wat is daar nu de zin van? Welnu, subtypes erven alles van het Parent (of Super)type: alle attributen en alle members. Een leraar en leerling hebben ongetwijfeld een naam en geboortedatum, maar dat hebben alle personen (nemen we in ons model aan). Daarom zullen de attributen 'naam' en 'geboortedatum' gedefinieerd worden bij Object Type 'persoon'. 'Leraar' en 'leerling' erven deze en het is derhalve niet nodig ze op Subtype niveau alsnog te definiëren. Op 'leraar' niveau kan bijvoorbeeld een additioneel attribuut als 'vak' (het vak dat de leraar mag onderwijzen) en op 'leerling' niveau een attribuut als 'klasnummer' gedefinieerd worden. Beide attributen zouden onzinnig zijn om op persoonniveau bij te houden. Hetzelfde gaat op voor de members van een Object Type: 'geef_naam' is een typische persoon-member, maar 'geef_klasnummer' een typische leerling-member.

## Nieuwe statements en clauses

Een reeks aan nieuwe statements en clauses komt in Oracle 9i tevoorschijn op het gebied van overerving: CREATE (OR REPLACE) TYPE [naam] UNDER [naam], OVERRIDING, INSTANTIABLE, FINAL, TREAT en SUBSTITUTABLE. Oracle biedt ondersteuning voor instantieerbare en niet instantieerbare (abstracte) Object Types (INSTANTIABLE), biedt de mogelijkheid om aan te geven of een Type wel of geen subtypes kan hebben (FINAL) en

of een member niet of wel anders geïmplementeerd mag worden in de subtypes (OVERRIDING en FINAL). Tevens biedt het mogelijkheden om een instantie van een supertype te 'behandelen als' (TREAT) een type van een Subtype en om te verhinderen dat een Subtype aan een variabele/kolom van het Supertype toegewezen mag worden (SUBSTITUTABLE). Oracle ondersteunt zogeheten 'Single inheritance', ofwel een Subtype heeft altijd precies één Parent type.

Polymorfie (Engels: polymorphism), ofwel de capaciteit van de OO-taal om te kunnen kiezen uit meerdere methodes van dezelfde naam en de goede te kiezen, is voor het eerst in dynamische vorm aanwezig in Oracle9i als gevolg van overerving. Statische polymorfie kenden we in eerdere versies van PL/SQL al in het geval van bijvoorbeeld packages die overloaded versies van procedures en functies konden bevatten en waartussen Oracle op compile tijd een keuze maakte. Dynamische polymorfie is nieuw in 9i en maakt het mogelijk voor Oracle om op runtime te besluiten welke methode te gebruiken, door de methode te kiezen geïmplementeerd in het Object Type dat 'het dichtst bij' is in de hiërarchie. Een voorbeeld, waarmee de Oracle9i syntax van het definiëren van Object Types eveneens wordt getoond, zal dit duidelijk maken:

```
CREATE OR REPLACE TYPE persoon_ot IS OBJECT (
   naam              VARCHAR2 (100),
   geboortedatum     TIMESTAMP (3),

   MEMBER FUNCTION toon_leeftijd RETURN INTERVAL YEAR TO MONTH,
   FINAL MEMBER FUNCTION toon_naam RETURN VARCHAR2,
   MEMBER PROCEDURE nieuwe_naam (naam_in IN VARCHAR2)
)
INSTANTIABLE
NOT FINAL;

CREATE OR REPLACE TYPE leraar_ot UNDER persoon_ot (
   vak VARCHAR2(100),
   FINAL MEMBER FUNCTION toon_vak RETURN VARCHAR2,
   OVERRIDING MEMBER PROCEDURE nieuwe_naam (naam_in IN VARCHAR2)
)
INSTANTIABLE
FINAL;

CREATE TABLE mens (zomaar_iemand persoon_ot)
COLUMN zomaar_iemand NOT SUBSTITUTABLE AT ALL LEVELS;

INSERT INTO mens
     VALUES (persoon_ot (
                'Toine van Beckhoven',
                '01-JAN-70'
             )
             )
```

Bovenstaande code geeft het volgende weer: twee definities van Object types (de implementatie gebeurt vervolgens via een CREATE OR REPLACE TYPE BODY, niet weergegeven hier), persoon_ot en leraar_ot. Leraar_ot is een Subtype van persoon_ot. Leraar_ot erft daardoor twee attributen (naam en leeftijd), drie members (toon_leeftijd, toon_naam en nieuwe_naam, implementeert zelf (override) de nieuwe_naam procedure opnieuw en is de laatste in de hiërarchie (er mag geen Subtype van leraar_ot gemaakt worden). Van beide types mogen instanties gemaakt worden.
Het 'dynamische polymorfie'-mechanisme betekent nu, dat in geval de methode 'nieuwe_naam' van een instantie van persoon_ot wordt aangeroepen, PL/SQL de implementatie neemt zoals gedefinieerd in het persoon_ot Type. Maar als deze methode wordt aangeroepen van een instantie van leraar_ot, dan pakt het de implementatie van leraar_ot. Stel nu, dat leerling een Subtype zou zijn van persoon en op leerling niveau wordt deze methode NIET opnieuw geimplementeerd, dan pakt PL/SQL de implementatie van de methode het dichtst bij in de hiërarchie en dat is op persoon niveau. Tenslotte wordt een tabel gecreëerd met daarin één kolom van het type persoon_ot en waarvan gespecificeerd wordt dat we niet willen dat er een Subtype van persoon_ot in die kolom geplaatst kan worden. Standaard kan dat namelijk wel: zonder de regel 'NOT SUBSTITUTABLE AT ALL LEVELS'

```
INSERT INTO mens
     VALUES (leraar_ot (
                'Toine van Beckhoven',
                '01-JAN-70',
                'Bestuurlijke informatiekunde'
             )
             )
```

kan het volgende statement succesvol uitgevoerd worden:
In Oracle9i is de Object Option aldus voorzien van een belangrijke OO-eigenschap. Niettemin zal het gebruik ervan afhankelijk zijn van smaak. Object Oriëntatie vergt een geheel andere manier van denken dan Relationeel en Procedureel programmeren en met de Object Option kunnen heel eenvoudig zeer complexe constructies gemaakt worden (klinkt bijna paradoxaal). Niettemin zullen er zeker OO-liefhebbers zijn in de Oracle wereld die nu meer dan ooit reden zien om deze optie te gaan exploiteren.

## Nieuwe datatypes

Naast de uitbreidingen op het gebied van de Object Option, zijn met name ook de nieuwe ontwikkelingen op het gebied van nieuwe datatypes en de ermee samenhangende built-in functies en packages heel interessant. Versie 9 Release 2 wordt al wel de 'XDB' genoemd, wat staat voor 'XML Database' en dat is omdat de XML ondersteuning in 9i (en in Release 2 nog verder) sterk is uitgebreid. De duidelijkste is de toevoeging van een XML datatype, zodat variabelen en kolommen nu van het type XMLType gedeclareerd kunnen worden. Het XMLType is een Object Type en zoals we eerder zagen betekent dit dat er eveneens members aanwezig zijn om de instanties van het type mee te bewerken. De opslag van een waarde van XMLType is overigens van het type CLOB (Character Large Object). Er kunnen Xpath expressies, zowel vanuit SQL als PL/SQL gedaan worden op de inhoud van de XMLType kolom/variabele. Xpath is simpel gezegd de query taal van XML, vergelijk het met wat SQL is voor een Relationele database. Een voorbeeld van het gebruik van dit datatype in een PL/SQL blok laat dit zien:

```
DECLARE
   my_xml    sys.xmltype;
BEGIN
   my_xml :=
         sys.xmltype.createxml (
            '<?xml version="1.0"?>
                <adres>
                   <straat>Willemll Straat</straat>
                <plaats>Kruikenzeikersstad</plaats>
                </adres>'
         );

   IF my_xml.EXISTSNODE ('/adres/straat') = 1
   THEN
      DBMS_OUTPUT.put_line ('Yes');
   ELSE
      DBMS_OUTPUT.put_line ('No');
   END IF;
END;

-- Output: Yes
```

De members CreateXML en ExistsNode zijn voorbeelden van members van het datatype XMLType.

Naast het XMLType datatype zijn er een paar hele aparte datatypes in 9i toegevoegd, waarmee generiek programmeren nog eenvoudiger

wordt. Heeft u wel eens data van verschillende datatypes willen opslaan in één kolom? Dan gebruikte u wellicht iets van het type LONG RAW, of misschien een VARCHAR2, maar dan waren er toch heel wat beperkingen, want je kon nergens aan zien wat het datatype werkelijk was van de inhoud (of je moest dat weer opslaan in een aparte kolom).
Oracle 9i introduceert de zogeheten 'Any'-datatypes, ANYTYPE, ANYDATA en ANYDATASET! In een kolom van het type ANYDATA[SET] kunnen instanties van het type VARCHAR2, NUMBER, BLOB, BFILE, TIMESTAMP en dergelijke, ofwel alles wat in de nieuwe DBMS_TYPES package gespecificeerd staat, door elkaar worden opgeslagen en Oracle weet dan nog wat het is ook! Ook voor deze datatypes geldt dat het Object Types zijn, en ook zij hebben derhalve members om de data mee te bewerken, waarmee conversies gedaan kunnen worden en het datatype uitgevraagd kan worden. Ongekende mogelijkheden!

Het nieuwe TIMESTAMP datatype maakt het mogelijk om een grotere precisie van secondes op te slaan en ondersteunt ook tijdzone verschillen. Naast een TIMESTAMP kunnen met het INTERVAL datatype ook periodes opgeslagen worden. INTERVAL DAY TO SECOND (de meest nauwkeurige) en INTERVAL YEAR TO MONTH slaan periodes op in respectievelijk dagen-seconden en jaren-maanden.

## Table functies en cursor expressies

In Oracle8i bestond al de mogelijkheid om TABLE functies te gebruiken in de FROM-clause van een SQL query: met TABLE functies kunnen resultaat sets die als RETURN waarde van een functie worden teruggegeven, worden gebruikt als ware het tabellen/views in de database. Dat ziet er ongeveer zo uit, waarbij de functie 'my_table_function' een stored function is die een collectie teruggeeft (bv. een TABLE of OBJECT, in het voorbeeld my_object_tab):

```
SELECT * FROM TABLE ( CAST ( my_table_function() AS my_object_tab ) )
```

Nieuw in 9i is de mogelijkheid om als argument aan de TABLE function een REF cursor mee te geven, waardoor de mogelijkheid geschapen wordt om de functie als een 'transformatie' functie te gebruiken: het doorgeven van resultaat sets van de ene aan de andere functie zonder de noodzaak van zelf te definiëren tussenresultaten zoals tijdelijke of fysieke tabellen. De voordelen daarvan zijn toegenomen performance (ondersteuning voor parallelle processing) en beter verbergen van complexe logica. TABLE functies kunnen in 9i 'gepipelined' worden, zodat rijen al van de ene aan de andere functie doorgegeven kunnen worden voordat de hele functie gereed is! Met name in Data Warehouse omgevingen zullen deze mogelijkheden interessant zijn.

Cursor expressies waren ook in Oracle8i beschikbaar, maar alleen in SQL. In 9i zijn ze te gebruiken in PL/SQL. Cursor expressies zijn cursor declaraties waarbij in het SELECT statement een kolom op zichzelf weer een cursor is. Hiermee kunnen geneste cursors gedeclareerd worden, zodat in één query resultaat sets van meerdere tabellen opgevraagd kunnen worden en cursor open/close management vereenvoudigd wordt. Cursor expressies retourneren derhalve geneste cursors.

```
CURSOR my_parent_cur IS
SELECT col_name,
   CURSOR (SELECT col_name,
              CURSOR(SELECT col_name...)...
```

In 9i kan een CURSOR expressie als argument aan een TABLE functie meegegeven worden (vergelijk met de 'REF Cursor' eerder genoemd als de mogelijkheid om transformatie TABLE functies te maken).

## Overige nieuwe functionaliteit

Veel van de nieuwe functionaliteit in PL/SQL 9i is heel krachtig, maar ziet er ook angstaanjagend uit, wat de adoptie ervan waarschijnlijk niet ten goede komt. Multi-level collecties is er ook zo één: Oracle9i maakt het mogelijk om collecties te nesten, dus collecties van collecties van collecties van collecties enzovoort. Collecties zijn in Oracle Index-by-tables, Nested Tables en Varrying Arrays. Ofwel je kunt multidimensionele arrays gaan creëren! Maar dat leidt ook tot code als:

```
My_index_by_table_variable(1005)(111)(2000)
:= '2';
```

als index voor een drie niveaus diep geneste Index-by-table bijvoorbeeld. Dit is exact de reden dat Feuerstein zijn presentatie eigenlijk 'The Wild and Whacky World of 9i PL/SQL' wilde noemen: 'wild' en 'idioot', want het wordt wel heel onleesbaar. Niettemin, voor degenen die het verstandig gebruiken prachtige opties.

Als laatste van de 9i new features noemde Feuerstein nog de volledige ondersteuning voor het CASE statement, want naast de Simple Case in 8i is er nu ook de Case expressie en de ondersteuning ervan in PL/SQL.

## Resumerend

Feuerstein liet ons in zijn bekende snelle tempo een selectie van 9i PL/SQL nieuwe features zien, die ons alledaagse leven als programmeur kunnen verrijken (of beangstigen). Op de website van Quest (http://www.questpipelines.com/Pipelines/quest_experts.htm) staan alle gebruikte voorbeelden en de complete presentatie van Feuerstein, waarvan het Oracle 9i nieuwe features deel slechts het topje van de ijsberg is. ■

*Toine van Beckhoven*
*Motiv IT Masters*