Presentatie PL/SQL expert Steven Feuerstein voor OGh-leden

The Magic and Mystery of PL/SQL Collections

Op 5 november jl. heeft de Amerikaan Steven Feuerstein zijn inmiddels bijna 'traditionele' presentatie voor OGh-leden gehouden. De belangstelling voor deze bijeenkomst bij Oracle in De Meern was overweldigend, waardoor meerdere inschrijvers teleurgesteld moesten worden. Feuerstein trakteerde de OGh-leden op een avond PL/SQL collections en bracht ze in zijn bekende snelle tempo de redenen bij waarom ze deze PL/SQL datastructuren zouden kunnen (of moeten) gebruiken.

Voor de vierde maal sprak de man die wordt beschouwd als 's werelds grootste expert in Oracle's procedurele taal, PL/SQL. In 1999 gaf hij een uiteenzetting over de '10 domste dingen ooit gezien of gedaan met PL/SQL'. In 2001 was daar de introductie van een Unit testing framework voor PL/SQL, gebaseerd op de principes van eXtreme Programming (XP) en het reeds bestaande unit testing framework voor Java, Junit. Dit framework werd door Feuerstein utPLSQL gedoopt en is op dit moment een al door velen gebruikte tool om gestructureerd PL/SQL te testen. Vorig jaar leidde Feuerstein OGh-leden in in de nieuwe features van Oracle 9i PL/SQL, waarbij en passant ook nog snel een reeks 8i 'new' features aan de orde kwam.
Elk jaar wordt weer duidelijk dat Feuerstein in zijn kennis van de PL/SQL taal en de noviteiten die Oracle introduceert een stuk verder is dan de gemiddelde aanwezige ontwikkelaar, want menig verbaasd of niet-begrijpend gezicht valt te bespeuren. En dat geldt in zekere zin ook voor het onderwerp van dit jaar, dat enigszins voortborduurt op dat van 2002: 'het magische en mysterieuze van PL/SQL collections'. Aan het einde van de presentatie werd ook SWYG geïntroduceerd: een nieuwe tool die het leven van de PL/SQL ontwikkelaar verder zal moeten veraangenamen en waarvan binnenkort een eerste werkende versie zal verschijnen.

PL/SQL collections

Sinds Oracle7 kent iedere PL/SQL ontwikkelaar wel de zogeheten 'PL/SQL tables': datastructuren waarin in het gebruikersgeheugen een lijst gemaakt kan worden, geïndexeerd via een BINARY_INTEGER en waarvan elke entry kan bestaan uit een waarde van één datatype, dan wel uit een record. Deze PL/SQL tabellen lenen zich prima voor snelle toegang in geheugen tot (lijsten van) gegevens. Ze zijn waardevol gebleken (en nog steeds) voor het cachen van gegevens uit de database in geheugen, in workarounds voor het mutating table probleem in database triggers en als emulatie van bi-directionele cursors (voorwaarts, maar ook terugwaarts door een resultaatset navigeren), die PL/SQL standaard niet kent.

Sinds Oracle8 zijn naast PL/SQL tables, die in Oracle8 werden omgedoopt tot 'index-by tables', nog twee collectie soorten geïntroduceerd: VARRAY's en NESTED TABLE's. VARRAY's zijn beschikbaar in SQL en PL/SQL en zijn bedoeld voor kleine lijstjes, op te slaan bijvoorbeeld in een kolom van een relationele tabel. VARRAY's hebben een (te definiëren) maximum aantal entries en de volgorde waarin de elementen in een variabele of kolom van type VARRAY worden opgeslagen is van belang. Nested tables (NT) daarentegen hebben geen maximum aantal entries (behoudens afhankelijk van de hoeveelheid intern geheugen) en de volgorde van de elementen is niet van belang: NT's zijn zogenoemde 'multisets', een set waarin de volgorde genegeerd wordt, maar dubbele entries wel degelijk bewaard blijven. Multisets {a,b,c} en {b,c,a} zijn hetzelfde, maar {a,b,c} en {a,a,b,c} niet!
In hun eerste implementatie in Oracle8 konden VARRAY's en Nested Tables nog niet genest worden. In Oracle9 kan dit wel. De volgende code geeft een voorbeeld van het nesten van een Nested Table, in dit geval van het nesten van een 'collectie' onderdelen in een collectie 'wedstrijden'.

```
1 CREATE OR REPLACE TYPE onderdeel_nt AS
  TABLE OF VARCHAR2(30);
2 CREATE OR REPLACE TYPE wedstrijd_nt
  AS TABLE OF onderdeel_nt;
3 DECLARE
4 v_wed wedstrijd_nt :=
5 wedstrijd_nt (onderdeel_nt ('100 meter'), onder-
  deel_nt ('verspringen'));
6 BEGIN
7 DBMS_OUTPUT.put_line (v_wed (2) (1));
8 END;

Output: verspringen
```

In deze listing worden direct wat constructies duidelijk die anders zijn dan bij het werken met 'Oracle 7' PL/SQL tables: het feit dat er een collection type gedefinieerd kan worden van een ander collection type (regel 2), het instantieren van de Nested table variabele via een Constructor (met dezelfde naam als het TYPE) (regels 4 en 5) en het opvragen van een waarde uit een 'cell', in dit geval de cel op positie (2)(1). De volgorde van de indices is wellicht andersom dan verwacht: de 'buitenste' collectie index staat achteraan en de 'binnenste' staat vooraan. Dat zijn zo de mysterieuze aspecten van collections waar Feuerstein zijn presentatie naar vernoemde. Overigens zijn collecties van records of objecten niet zo anoniem: deze hebben een naam.

Door de mogelijkheid van geneste collections wordt het heel mooi mogelijk om complexe datastructuren waarin een hiërarchie zit, te vereenvoudigen in PL/SQL variabelen. Feuerstein gaf in dit verband het voorbeeld van de data dictionary tabel ALL_ARGUMENTS, dat hij in combinatie met built-in package DBMS_DESCRIBE heeft moeten gebruiken in ene utility die hij onlangs heeft gemaakt, 'Codecheck': een code analyse tool, met als eerste doel het detecteren van ongeldige overloadings in packages (en die de compiler niet opmerkt). Feuerstein heeft hierover op Oracle Technology Network (OTN) een reeks artikelen geschreven. Door gebruik van Nested Collections wist hij de ingewikkelde informatie (in een hiërarchische structuur) in de platgeslagen tabel ALL_ARGUMENTS vele malen bruikbaarder te maken.

Oracle 10g

In Oracle 10g introduceert Oracle enkele nieuwe operators die collections kunnen bewerken: MULTISET UNION, MULTISET UNION DISTINCT, MULTISET EXCEPT, MULTISET INTERSECT en SET. Omdat Nested Tables -zoals boven aangegeven - verzamelingen zijn waarbij dubbele entries wel significant zijn en volgorde niet, verschilt MULTISET UNION (waarbij twee collecties bij elkaar gevoegd worden) van de SQL operator UNION: De collection's MULTISET UNION verwijdert geen duplicaten, terwijl SQL's UNION dit wel doet. Voor collections die bij elkaar gevoegd worden en waarvan duplicaten verwijderd moeten worden, is er de MULTISET UNION DISTINCT. De MULTISET EXCEPT en INTERSECT komen overeen met SQL's MINUS en INTERSECT. De collection operator SET tenslotte verwijdert duplicaten uit een collectie.

Performance

Het in Oracle8i geïntroduceerde Bulk processing van collecties heeft gezorgd voor spectaculaire performance verbeteringen in het gebruik van collections in combinatie met SQL statements. De FORALL en BULK COLLECT statements reduceren namelijk het aantal context switches tussen de PL/SQL engine en de SQL engine die Oracle moet ondergaan als vanuit PL/SQL een SQL statement wordt uitgevoerd. Een eenvoudige en ogenschijnlijk onschuldige loop als:

```
BEGIN
FOR i IN onderdelentab.FIRST..onderdelentab.LAST LOOP
UPDATE wedstrijd_onderdelen
SET onderdeel = onderdelentab(i).onderdeel
WHERE onderdeel_id = onderdelentab(i).onderdeel_id;
END LOOP;
END;
```

Is, in het geval de 'onderdelen' tabel ettelijke honderden of duizenden records bevat, goed voor even zovele context switches, die relatief duur zijn in Oracle. Aangenomen dat een simpel SQL update statement in bovenstaand geval niet voldoet, omdat er bijvoorbeeld nog allerlei ingewikkelde transformaties moeten plaatsvinden voor elk record en de ontwikkelaar genoodzaakt is tot een loop door de records, is een FORALL statement hier vele malen sneller:

```
BEGIN
FORALL i IN onderdelen_tab.FIRST .. onderdelen_tab.LAST
UPDATE wedstrijd_onderdelen
SET onderdeel = onderdelen_tab (i)
WHERE onderdeel_id = onderdelenid_tab (i);
END;
```

Merk op dat in geval van FORALL (en ook BULK COLLECT) in Oracle8i geen collections van records mogelijk zijn, vandaar de twee aparte collections in bovenstaand voorbeeld. In Oracle9i is deze beperking opgeheven, wat het gebruik van bulk processing vele malen eenvoudiger maakt. En mogelijk de grootste verbetering is het kunnen samenstellen van dynamisch SQL in combinatie met FORALL en BULK COLLECT, wat voor een bijna ongelimiteerde flexibiliteit zorgt zonder toe te geven op performance! Zie bijvoorbeeld het volgende statement:

```
BEGIN
FORALL i IN onderdelen_tab.FIRST .. onderdelen_tab.LAST
EXECUTE IMMEDIATE
'UPDATE wedstrijd_onderdelen
SET onderdeel = REPLACE(onderdeel,'||str_in||')
WHERE onderdeel_id = :1
RETURNING onderdeel INTO :2'
USING onderdelenid_tab (i)
RETURNING BULK COLLECT INTO wedstrijdonderdelentab;
END;
```

Het aantal redenen om geen Bulk processing te gebruiken is drastisch gereduceerd met deze mogelijkheden. Redenen als complexe processing per rij of geheugenlimieten (een collection staat in user memory en is niet geshared) of de behoefte meer controle te houden over het proces, kunnen redenen zijn om de conventionele LOOP's te gebruiken in plaats van FORALL of BULK COLLECT statements. Maar in veel gevallen kan bulk processing een enorme performanceverbetering opleveren in de orde van grootte van honderden of zelfs duizenden malen sneller. En dat zijn getallen die het het proberen waard maken. Fouten die optreden tijdens bulk processing kunnen in Oracle 9i nu ook bewaard worden in een pseudo collection die na het bulk proces uitgelezen en eventueel afgehandeld kan worden. Deze pseudo collectie heet SQL%BULK_EXCEPTIONS en wordt gevuld middels het statement SAVE EXCEPTIONS in een bulk process.

SWYG

Aan het einde van de presentatie werden de aanwezigen nog getrakteerd op een onvervalst stukje marketing: Feuerstein vertelde enthousiast over SWYG, dat staat voor 'Show Me What (you) Got', een tool die hij momenteel samen met anderen bouwt. SWYG is bedoeld om een brug te slaan tussen de low-level IDE's die momenteel bestaan (en die zonder twijfel voor veel productiviteitsverhoging zorgen) en de denkwereld van een ontwikkelaar bij het oplossen van een probleem: het denken in hogere niveau concepten en entititeiten in plaats van in tabellen, functies en procedures. SWYG is geen vervanger van IDE's, maar een complementair product, waarin code generatie een belangrijke rol speelt. Onder aan dit artikel staat een link naar de website over SWYG, waar meer informatie te lezen is en waar SWYG, zodra er een werkende versie gereed is, gedownload kan worden.

Tot slot

Het was wederom een leerzame PL/SQL avond, waarbij Feuerstein de aanwezige OGh-leden opnieuw wist te overtuigen dat ondanks de komst van Java enkele jaren geleden, PL/SQL nog altijd de te prefereren procedurele taal is en dat Oracle niet van plan lijkt PL/SQL te laten doodbloeden. Integendeel: Oracle 8i, 9i en nu weer 10g laten vele verbeteringen zien in zowel functionaliteit en performance, waardoor PL/SQL meer leeft dan ooit. Steven Feuerstein is daarom voorlopig nog niet uitgesproken en uitgeschreven en er is grote kans dat we hem volgend jaar weer kunnen aanschouwen in Nederland. ■

Toine van Beckhoven, Motiv IT masters

Links:

'Feuerstein Feuerstein': http://www.Feuersteinfeuerstein.com, met verwijzingen naar presentaties en voorbeeldcode (alle vrij te downloaden) 'utPLSQL': http://utplsql.sourceforge.net
'Taking up collections' (Oracle 10g verrijkingen):
http://otn.oracle.com/oramag/oracle/03-sep/o53plsql.html
'Programming at multiple levels':
http://otn.oracle.com/oramag/oracle/02-may/o32plsql.html
'SWYG': http://www.swyg.com