PL/SQL-sessie

ciber
ALWAYS ABLE

Rob van Wijk

19 januari 2009

Inleiding

Deze PL/SQL-sessie is bedoeld voor ervaren programmeurs in andere talen dan PL/SQL, die tevens al kennis hebben van SQL. Hierdoor zal deze sessie geen algemene programmeeroefeningen of SQL-oefeningen bevatten, maar alleen de PL/SQL concepten, commando's en syntax behandelen.

De onderwerpen zijn:

1. Positionering, architectuur en codeopbouw
2. Datatypen
3. Belangrijkste commando's
4. Foutafhandeling
5. Verschijningsvormen van PL/SQL
6. SQL in PL/SQL
7. Cursoren
8. Collecties en recordtypes
9. Bulkverwerking
10. Dynamisch SQL

Voorbereiding

Om een vliegende start te kunnen maken op de dag zelf, verzoek ik jullie het volgende thuis al te doen:

Download de Oracle-database versie 11.1.0.6 vanaf http://www.oracle.com/technology/software/products/database/index.html en installeer de database.

Tip: de Oracle-processen worden standaard opgestart als je laptop opstart. Dit kan je veranderen door in de Windows "Control Panel" --> Administrative Tools --> Services deze services op "Manual" te zetten.

Maak een startora11.bat met de inhoud:
net start <naam listener-service>
net start <naam database-service>

Op dezelfde manier maak je ook een stopora11.bat
net stop <naam listener-service>
net stop <naam database-service>

Door het aanklikken van deze bestandjes kan je nu gemakkelijk de database starten en stoppen.

De sessie is ontwikkeld om te werken met SQL*Plus. In versie 11 is er geen Windows-versie meer, dus ik heb het hier over de "DOS-versie" sqlplus.exe. De opgaven en antwoorden zullen hierop aansluiten. Om SQL*Plus een beetje fraai te configureren, kan ik jullie het volgende artikel aanbevelen: http://www.williamrobertson.net/documents/sqlplus-setup.html.

Start SQL*Plus en meld je aan met gebruikersnaam system en het tijdens de installatie gekozen wachtwoord. Maak een gebruiker aan met jouw eigen naam of eigen code ("create user rwijk identified by rwijk;") en ken de DBA rol aan deze gebruiker toe ("grant dba to rwijk;"). Het wordt afgeraden om onder de gebruikers SYS en SYSTEM te werken, dus vanaf nu kan je uit de voeten met je eigen gebruikerscode. Installeer de voorbeeldtabellen EMP en DEPT (en nog een paar andere tabellen) door het script demobld.sql te draaien. Dit script kan je vinden op internet door op "demobld.sql" te googlen.

1. Positionering, architectuur en codeopbouw

Dit zal aan de hand van een korte presentatie uitgelegd worden.

2. Datatypes

PL/SQL kent veel datatypes. De basis drie/vier zijn: NUMBER, VARCHAR2 en DATE/TIMESTAMP. Deze komen overeen met de meest voorkomende datatypes van kolommen in de database.

2.1 Datatype NUMBER

Het datatype NUMBER heeft twee parameters en zou dus eigenlijk zo geschreven moeten worden: NUMBER(precisie,schaal). De schaal geeft het aantal getallen rechts van de decimale komma aan. Voor integers is de schaal dus 0 en dit is ook de default waarde. Voor integers mag je dus ook NUMBER(precisie) gebruiken. Parameter precisie is een getal tussen 1 en 38, waarbij 38 de default waarde is. Precisie geeft het totaal aantal getallen weer, inclusief diegene achter de decimale komma. Een NUMBER is dus eigenlijk een NUMBER(38,0).

a) Draai pls21a.sql en probeer de uitvoer te begrijpen.

b) Maak zelf een identiek stukje code als in pls21a.sql, maar declareer de variabele dan als NUMBER(2,4) en achterhaal proefondervindelijk welke waarden deze variabele kan bevatten.

2.2 Datatype VARCHAR2

Het datatype VARCHAR2 heeft maar 1 parameter die de maximale lengte weergeeft. Het bereik van deze parameter in PL/SQL loopt van 1 t/m 32767. In SQL is dat 1 t/m 4000. Een ouder datatype CHAR heeft de eigenschap dat strings werden aangevuld met spaties tot aan de maximale lengte. Dit leidt tot allerlei weliswaar verklaarbare maar toch verwarrende situaties. Voor strings wordt sinds jaar en dag eigenlijk alleen de VARCHAR2 gebruikt. Strings zelf staan tussen enkele quotes.

a) Draai pls22a.sql en bekijk het verschil in gedrag tussen een CHAR en een VARCHAR2.

2.3 Datatypes DATE en TIMESTAMP

Oracle heeft zijn eigen datatypes voor datums. Belangrijk is het besef dat een datum geen formaat heeft. Pas als je een datum converteert naar tekst, krijgt het een formaat. Lang was DATE het enige datatype voor datums, maar vanaf Oracle versie 9 is daar TIMESTAMP bijgekomen. Een DATE bevat componenten voor eeuw, jaar, maand, dag, uur, minuten en seconden. Een TIMESTAMP heeft een grotere precisie. Dit is afhankelijk van wat het besturingssysteem aankan. Zo gaat TIMESTAMP in Windows tot milliseconden en in UNIX tot microseconden. Maximaal mogelijke precisie is nanoseconden. Van TIMESTAMP zijn er varianten waarbij de tijdzone gespecificeerd kan worden.

a) De voorgedefinieerde SYSDATE en SYSTIMESTAMP geven de huidige datum en tijd aan. Ze zijn van het type DATE respectievelijk TIMESTAMP en worden in pls23a.sql gebruikt om de TO_CHAR functie te demonstreren. Met de TO_CHAR functie wordt het datatype geconverteerd naar een VARCHAR2. Bekijk daarna zelf in de SQL Reference Manual welke datum formaatelementen er zoal zijn.

b) Om een DATE variabele zelf een waarde te geven, moet eerst een string naar een date worden geconverteerd. Bekijk in de SQL Reference Manual de TO_DATE functie en schrijf code die de waarde 19 januari 2009 14:50 uur toekent aan een DATE variabele. Gebruik

vervolgens de TO_CHAR functie om exact deze string af te drukken: '19 januari 2009 14:50u'.

c) Je kunt aan een datumvariabele ook een string toekennen. Oracle converteert de string dan impliciet naar een datum, indien mogelijk. Dit is echter sterk af te raden. Jouw code kan door het wijzigen van de omgevingsvariabele NLS_DATE_FORMAT zomaar omvallen en ook wordt de code stukken onleesbaarder en daardoor lastiger onderhoudbaar. Draai pls23c.sql voor een huiveringwekkend voorbeeld. Gebruik dus altijd de TO_DATE (of TO_TIMESTAMP) functie voor datums. Of gebruik de verkorte notatie uit 2.3d.

d) Met de TO_DATE en TO_TIMESTAMP functies heb je alle vrijheid om een datum of datumtijd in ieder gewenst formaat aan te leveren. Er is ook een kortere, door ANSI voorgeschreven manier om een datum of een datumtijd te specificeren. Je zit dan wel vast aan 1 voorgeschreven formaat. Voor een datum zonder tijd is dat: YYYY-MM-DD en voor een datum inclusief tijd is dat: YYYY-MM-DD HH24:MI:SS. Draai pls23d.sql voor een demonstratie.

2.4 Datatype BOOLEAN

SQL zelf kent geen boolean, maar PL/SQL wel. In PL/SQL kent een boolean geen twee mogelijke waarden, maar drie: TRUE, FALSE en NULL. Een NULL is een speciale waarde die als "onbekend" geïnterpreteerd kan worden en die in het begin nogal eens tot verwarring kan leiden. Zo goed als iedere expressie die ergens een NULL bevat, is gelijk aan NULL.

a) Draai het script pls24a.sql om te zien hoe de waarde van een boolean variabele uitgevraagd kan worden. Let daarbij vooral op in welke tak van het IF-statement de NULL waarde schiet.

b) Achterhaal met het laatste stukje code van pls24a.sql in het achterhoofd de waarheidstabel van de AND-operator.

AND	TRUE	FALSE	NULL
TRUE			
FALSE			
NULL			

c) Doe hetzelfde maar dan met de OR-operator.

OR	TRUE	FALSE	NULL
TRUE			
FALSE			
NULL			

2.5 Overige datatypes

Lees hoofdstuk 3, 3-1 t/m 3-21 vluchtig door om een idee te krijgen welke datatypes nog meer in PL/SQL zijn te gebruiken.

3. Belangrijkste commando's

3.1 Toekenning

Je kent waarden toe aan een variabele door de toekenningsoperator *":="*. PL/SQL kent dus een duidelijk onderscheid tussen de vergelijkingsoperator en de toekenning.

x = 1 Dit is een boolean expressie. Spreek uit als "x is 1".
x := 1 Dit is een toekenning. Spreek uit als "x wordt 1".

3.2 Conditionele commando's

PL/SQL kent twee conditionele commando's: IF en CASE.

a) Lees pagina's 4-2 t/m 4-8 in de PL/SQL Reference Manual tot aan de paragraaf "Controlling Loop Iterations" over de IF en CASE commando's.

b) Schrijf een programmaatje met het CASE-commando om te zien welke ORA-melding er wordt opgeworpen als de conditie niet in de lijst voorkomt (de CASE_NOT_FOUND exceptie).

c) Er is een verschil tussen het CASE-commando en de CASE-expressie. De CASE-expressie kan zowel in SQL als in PL/SQL gebruikt worden. Lees op pagina 2-40 t/m 2-42 in de PL/SQL Reference Manual over de CASE-expressie. Welk verschil in syntax met het CASE-commando valt je op? Wat is de waarde van een CASE-expressie als de conditie niet in de lijst voorkomt?

3.3 Iteratieve commando's

PL/SQL kent vier soorten iteratieve commando's: LOOP, WHILE-LOOP, FOR-LOOP en de cursor-FOR-LOOP. De laatste zal in hoofdstuk 7 over cursoren worden behandeld.

a) Lees de pagina's 4-8 t/m 4-20 door uit de PL/SQL Reference Manual.

b) Schrijf een programma dat de getallen 1 t/m 10 toont met een dbms_output.put_line procedure. Doe dit met een basis LOOP, een WHILE-LOOP en met een FOR-LOOP.

4. Foutafhandeling

Foutafhandeling in PL/SQL doe je met excepties. Niet alle fouten wil je afvangen: excepties zijn je vriend en tonen je precies wat er waar is foutgegaan. Vang dus alleen fouten af die je verwacht en waar je je programma op wilt laten anticiperen. Zonder exceptie-sectie in je code "propageren" fouten naar een niveautje hoger, de aanroeper. Met een exceptie-sectie kun je ze opvangen en afhandelen of toch doorgeven naar boven met een RAISE-commando. De laatste (zonder parameter) werkt alleen in de exceptie-sectie, niet in het reguliere programmagedeelte.

4.1 Fouten opwerpen

Eigen applicatiefouten werp je op met RAISE_APPLICATION_ERROR.

a) Lees pagina 11-8 in de PL/SQL Reference Manual over de RAISE_APPLICATION_ERROR procedure.

Duizend foutmeldingen blijkt in de praktijk voor een applicatie vaak niet toereikend en daarom zie je vaak de volgende methode: er wordt een foutmeldingen tabel aangemaakt met minimaal de kolommen foutcode en fouttekst. De applicatie doet bij een foutsituatie een RAISE_APPLICATION_ERROR(-20000,'APP-nnnnn'). De applicatie bevat een generieke routine die met de tekst APP-nnnnn de foutmeldingen-tabel bevraagt naar de echte foutmelding. De laatste wordt dan aan de gebruiker getoond.

4.2 Voorgedefinieerde excepties

PL/SQL heeft een aantal veel voorkomende Oracle-fouten een eigen exceptienaam gegeven.

a) Bekijk de paragraaf "Predefined PL/SQL Exceptions" vanaf pagina 11-4 in de PL/SQL Reference Manual voor een lijst met de voorgedefinieerde excepties.

b) Schrijf een programma dat een x en een y variabele declareert als NUMBER en vervolgens het resultaat van de expressie x/y via dbms_output.put_line toont. Als y een 0 bevat, dan moet het programma geen fout retourneren, maar de tekst 'Delen door nul is flauwekul.' tonen.

4.3 Eigen excepties

Je kunt eigen excepties declareren. In de declaratiesectie declareer je een daartoe een variabele van het type EXCEPTION:

<je eigen exceptienaam> EXCEPTION;

In je programma kan je deze exceptie opwerpen door een:

RAISE <je eigen exceptienaam>;

Programmatechnisch is dit niet zo spannend: er zijn andere manieren om hetzelfde te bereiken. Echter is deze manier wel zelfdocumenterend als je een goede naam kiest voor je exceptie, en daarom nuttig.

a) Breidt het programma van 4.2b uit, zodanig dat als de x variabele een negatief getal bevat, een eigen exceptie wordt opgeworpen met de naam e_negatieve_teller. Vang in de exceptie-sectie deze eigen exceptie op en toon de tekst 'Negatieve teller'.

4.4 Afvangen overige ORA-fouten

Lees de paragraaf "Associating a PL/SQL Exception with a Number (EXCEPTION_INIT Pragma)" in de PL/SQL Reference Manual op pagina 11-7.

a) Schrijf een programma dat een string van het formaat dd-mon-yyyy naar een datum converteert. Als de string een ongeldige maand bevat, bijvoorbeeld '01-ABC-2009', dan wordt de datum gelijk aan 1 januari 2000.

4.5 WHEN OTHERS

De WHEN OTHERS vangt alle foutmeldingen op. Deze kan handig zijn, bijvoorbeeld als je alle foutmeldingen wilt loggen in een aparte tabel. Zorg er dan wel voor dat je de fout weer terug opwerpt met een RAISE commando. Vergeet je dit, dan denkt de aanroeper dat alles goed is gegaan, terwijl de vreselijkste dingen gebeurd kunnen zijn. Een WHEN OTHERS zonder een RAISE commando is dus eigenlijk altijd een bug in je programma! Als in regel 1 van jouw programma iets fout gaat, schiet de code in de WHEN OTHERS, die vervolgens voor de buitenwereld doet of alles goed is gegaan. Je zegt dus eigenlijk dat het niet uitmaakt of jouw programma gedraaid heeft, of niet.

In de WHEN OTHERS kan je de SQLCODE uitvragen, die het ORA-foutnummer bevat. Dit is een alternatief voor de PRAGMA EXCEPTION_INIT uit de vorige paragraaf, maar wel een minder goed alternatief, aangezien het minder zelfdocumenterend is.

5. Verschijningsvormen van PL/SQL

Tot nu toe hebben we alleen PL/SQL gezien in zogenaamde anonieme blokken. Deze worden eigenlijk alleen gebruikt voor test- en demonstratiedoeleinden, of als onderdeel van een (batch-)script. Andere vormen zijn:

- Benoemde PL/SQL blokken
- Database functies
- Database procedures
- Database packages (specificatie en body)
- Database triggers
- Database types (specificatie en body)

5.1 Benoemde PL/SQL-blokken

Benoemde PL/SQL blokken zijn hetzelfde als anonieme blokken, met het verschil dat je het blok een naam geeft. Voor puristen erg leuk om de reikwijdte (Engels: scope) van een blok en variabelen beter inzichtelijk te krijgen.

a) Draai pls51a.sql om te zien hoe je een blok een naam geeft, en hoe je vervolgens deze naam kunt gebruiken.

Als je twee variabelen met dezelfde naam declareert op verschillende niveaus, dan kunnen benoemde PL/SQL blokken uitkomst bieden. In de praktijk zie je dit nauwelijks: je zorgt ervoor dat variabelen verschillende namen krijgen.

5.2 Database procedures

Een database procedure is een stuk PL/SQL code dat geïnterpreteerd of gecompileerd opgeslagen wordt in de database. Een procedure heeft nul of meer parameters. Een parameter is een:

- invoerparameter (IN, de default)
- zowel in- als een uitvoerparameter (IN OUT) of
- uitvoerparameters (OUT)

a) Lees in de PL/SQL Reference Manual pagina's 14-42 t/m 14-45 door voor de CREATE PROCEDURE syntax.

b) Maak een database procedure genaamd toon_kwadraat. De functie heeft 1 invoerparameter en toont (met dbms_output.put_line) het kwadraat van de invoerparameter.

5.3 Database functies

Een database functie is vergelijkbaar met een database procedure, alleen retourneert deze een waarde. Net als bij alle andere programmeertalen zijn functies met parameters van het type OUT of IN OUT afgeraden, maar het is technisch wel mogelijk. Wil je meer waarden teruggeven, gebruik dan procedures.

a) Lees in de PL/SQL Reference Manual pagina's 14-27 t/m 14-34 door voor de CREATE FUNCTION syntax. Richt je daarbij op het RETURN-gedeelte van de syntax en doe vooralsnog nog geen moeite alle geavanceerde extra mogelijkheden 100% te snappen.

b) Maak een functie genaamd kwadraat met 1 invoerparameter en een numerieke retourneerwaarde.

c) Pas de procedure uit 6.2.2 aan, zodat deze gebruik maakt van de functie kwadraat. Hint: let op het "OR REPLACE" gedeelte van de syntax.

5.4 Database packages

Een database package is een verzameling variabelen, procedures en functies die bij elkaar horen.

a) Lees pagina's 10-1 t/m 10-9 in de PL/SQL Reference Manual over packages.

b) Oefen de syntax voor een package door er één aan te maken met een publieke functie f, een publieke procedure p1, een privé procedure p2, een publieke globale variabele gv1 en een privé globale variabele gv2. De code in de functies en procedures maakt niet uit, dus gebruik het commando NULL; in de procedures en RETURN NULL; in de functie.

5.5 Database triggers

Een database trigger is een stuk code dat automatisch wordt gedraaid na een gebeurtenis in de database. Er zijn drie typen database triggers: DML-triggers, DDL-triggers en database operatie-triggers. De eerste categorie bestaat het langst en wordt ontzettend veel gebruikt in Oracle-systemen die ontwikkeld zijn vanaf Oracle versie 7 (rond 1995), met name voor het afdwingen van bedrijfsregels.

a) Lees pagina's 9-1 t/m 9-7 van de PL/SQL Reference Manual.

b) Maak een tweede DEPT tabel aan met de naam DEPT2, identiek qua structuur aan DEPT. Gebruik hiervoor het commando "create table dept2 as select * from dept where 1=0". Maak een database trigger aan die rijen in DEPT die vanaf nu worden toegevoegd, ook toevoegt aan DEPT2.

Extra) Implementeer de bedrijfsregel "Een werknemer mag niet meer verdienen dan zijn manager" met een database trigger.

6. SQL in PL/SQL

6.1 DML

DML staat voor Data Manipulation Language. Hieronder vallen in ieder geval alle INSERT, UPDATE, DELETE en MERGE commando's van SQL. Het fraaie van PL/SQL is dat deze SQL commando's naadloos zijn geïntegreerd in PL/SQL.

a) Schrijf een PL/SQL programma dat een werknemer aan de tabel EMP toevoegt met jouw eigen naam. Verzin voor de andere kolommen zelf waarden (of NULL).

b) Schrijf een PL/SQL programma dat de namen van alle werknemers in kleine letters zet met de LOWER functie. Toon vervolgens met dbms_output.put_line het aantal werknemers waarvan je de naam hebt gewijzigd. Hint: zoek in de PL/SQL Reference Manual naar de juiste functie hiervoor. Vergeet niet je programma te eindigen met een ROLLBACK-commando!

6.2 SELECT

SQL's SELECT commando is natuurlijk ook mogelijk in PL/SQL, alleen moet je specificeren wat je met de geselecteerde waarden wilt doen. Hiervoor voeg je tussen de SELECT en de FROM een INTO clausule toe. Achter de INTO specificeer je de variabelen waarin de selectie moet worden opgevangen.

a) Schrijf een programma dat de naam en het salaris van empno 7839 uit de tabel EMP ophaalt en toont via dbms_output.put_line.

6.3 Transactie sturingscommando's

Hieronder vallen de COMMIT, SAVEPOINT, ROLLBACK (TO SAVEPOINT) and SET TRANSACTION commando's. Ook deze zijn naadloos geïntegreerd in PL/SQL. Er valt veel voor te zeggen om transactiesturing in de praktijk aan de aanroepende omgeving over te laten, en deze dus weinig te gebruiken. Als je in jouw programma's transactiesturing toevoegt, dan is immers de kans klein geworden dat jouw programma geschikt is om aangeroepen te worden. Hierover zijn echter de meningen niet eensluidend.

6.4 DDL

DDL staat voor Data Definition Language. Hieronder vallen alle commando's die de structuur van de database veranderen, zoals CREATE, ALTER en DROP commando's. Deze kun je niet zondermeer opnemen in PL/SQL. PL/SQL gaat uit van een vaststaande structuur van de database. Echter kan je met een EXECUTE IMMEDIATE commando dit omzeilen. Wees er wel van bewust dat dit programma's die zo dynamisch de structuur van de database wijzigen, heel wat wenkbrauwen van collega's doet fronsen.

a) Schrijf een programma dat dynamisch een tabel aanmaakt in het huidige schema, met het commando "create table magweg (id number(4))". Zoek hiertoe de syntax van het EXECUTE IMMEDIATE commando op in de PL/SQL Reference Manual.

7. Cursoren

Een cursor is een naam voor een *private SQL area*, een onderdeel van je PGA geheugen. Hierin staan o.a. de waarden van bindingsvariabelen, toestandsinformatie en de *work areas* nodig voor de uitvoering van jouw SQL. Als PL/SQL-programmeur word je van deze implementatiedetails afgeschermd, maar met cursoren heb je toch volledige controle over hoe je met data omgaat. Een cursor kan je daarom ook zien als het programmeer-equivalent van een SQL-zoekvraag. Een cursor declareer je in je declaratiesectie. Cursoren kun je openen: je laat dan de database de zoekvraag ontleden (parse) en uitvoeren (execute). Je kunt de rijen waarnaar een geopende cursor wijst ophalen (fetch), één of meer tegelijk. En als je klaar bent sluit je de cursor, waardoor het geheugen weer wordt vrijgegeven. Ook alle SQL uit de vorige hoofdstuk gebruikt onder water impliciet cursoren in de private SQL area, maar in dit hoofdstuk gaan we ze expliciet gebruiken.

7.1 Expliciete cursoren

a) Lees pagina's 6-9 uit de PL/SQL Reference Manual over Explicit cursors tot aan pagina 6-12 over "Fetching bulk data with a cursor".

b) Schrijf een programma met een cursor en een WHILE-LOOP dat alle namen van de werknemers uit afdeling 20 (emp.deptno = 20) toont.

7.2 Cursor-for-loops

a) Lees pagina's 6-18 tot 6-19 uit de PL/SQL Reference Manual tot aan de paragraaf "Using subqueries".

b) Schrijf het zelfde programma als in 7.1b, maar nu met een SQL cursor-for-loop en een expliciete cursor-for-loop.

7.3 Ref cursoren

a) Lees pagina's 6-23 tot en met 6-27 uit de PL/SQL Reference Manual tot aan paragraaf "Using a Cursor Variable as a Host Variable".

b) Schrijf een database procedure met een numerieke invoerparameter genaamd p_variant. De procedure toont alle werknemersnamen, afdelingsnummers en salarissen uit EMP. Als p_variant gelijk is aan 1, dan worden alleen de werknemers uit afdeling 20 geselecteerd en als p_variant gelijk is aan twee, dan worden alleen de mensen die meer dan 2000 verdienen geselecteerd. Gebruik hiervoor een ref cursor.

8. Collecties en recordtypes

Vaak zijn collecties arrays van records: een simulatie van een tabel om tijdelijk tabelinhoud te kunnen opslaan. Daarom eerst een uitleg en oefening met records.

8.1 Recordtypes

a) Lees "Defining and Declaring Records" op bladzijde 5-31 van de PL/SQL Reference Manual tot aan halverwege 5-38.

b) Oefen zelf met de RECORD-syntax door er een te definiëren met de velden empno, ename en sal, zoals in de EMP-tabel. Declareer een variabele van dit type en ken hieraan eigen waarden toe.

8.2 Collecties

PL/SQL kent drie soorten collectietypes: associatieve arrays, nested tables en varrays.

a) Lees de bladzijden 5-1 t/m 5-5 in de PL/SQL Reference Manual door om een idee van de drie soorten collecties te krijgen.

b) Vul onderstaande code aan zodat alle drie de collecties *aa*, *nt* en *va* drie elementen 'APPEL', 'BANAAN' en 'CITROEN' bevatten:

```
declare
  type t_associatieve_array is table of varchar2(10) index by pls_integer;
  type t_nested_table is table of varchar2(10);
  type t_varray is varray(5) of varchar2(10);
  aa t_associatieve_array;
  nt t_nested_table;
  va t_varray;
begin
  ...
end;
```

c) Breidt de code uit zodat alle drie de collecties worden doorlopen en de elementen worden getoond met een dbms_output.put_line. Gebruik voor het doorlopen van de collecties drie verschillende manier: eenmaal met een FOR-lus en de COUNT-methode, eenmaal met een WHILE-lus en de methodes FIRST en NEXT, en eenmaal met een WHILE-lus en de methodes LAST en PRIOR. Welke manier van doorlopen van een collectie zou je wanneer willen gebruiken?

9. Bulkverwerking

In huidige databases staan veelal miljoenen rijen in de tabellen die vaak batchgewijs verwerkingen ondergaan. Idealiter doe je dit alleen met SQL, maar soms is vanwege de complexiteit PL/SQL wenselijker en flexibeler. Voor Oracle versie 8 gebeurde dit veelal met rij-voor-rij verwerking met bijvoorbeeld cursor-for-loops. Maar dit leidt tot veel zogenaamde contextswitches: wisselingen tussen de PL/SQL engine en de SQL engine. Een contextswitch zelf kost weinig tijd. Zeer veel contextswitches gaan wel aantikken. Iedere fetch is een database aanroep. Iedere INSERT … VALUES commando is een database aanroep. En als je dit in een lus zet, dan heb je heel veel database aanroepen. Daarom is bulkverwerking geïntroduceerd: om het aantal database rondgangen te minimaliseren als het met alleen SQL niet lukt.

9.1 Bulkselectie met impliciete cursoren

Bulkselectie met impliciete cursoren lijkt erg op de normale SELECT … INTO syntax, alleen selecteer je nu niet in een gewone variabele maar in een collectie, en gebruik je BULK COLLECT INTO.

a) Selecteer alle werknemersnamen in afdeling 20 via een bulkselectie met impliciete cursor en toon ze vervolgens allemaal in een lus. Zie voor een voorbeeld pagina 12-17 paragraaf "Retrieving Query Results into Collections".

9.2 Bulkselectie met expliciete cursoren

Bulkselectie met expliciete cursoren lijkt ook enorm op een normale selectie met een expliciete cursor. Verschil is het FETCH commando dat nu FETCH … BULK COLLECT INTO … is, in plaats van FETCH … INTO … .

a) Doe hetzelfde als in 9.1a, maar dan met een expliciete cursor. Zie pagina 12-19 voor een voorbeeld.

Bulkselectie met expliciete cursoren is in de praktijk flexibeler, vanwege de mogelijkheid van de LIMIT-clausule. Als je een miljoen rijen met een bulkselectie in een collectie inleest, doe je een groot beroep op het PGA-geheugen en dit hindert de schaalbaarheid van je applicatie. Met de LIMIT-clausule kun je bulkselecties doen, maar in behapbare porties.

b) Haal weer de werknemersnamen uit afdeling 20 op, maar nu met een LIMIT van 2 rijen in een lus die 3 iteraties zal moeten uitvoeren. Zie voor een voorbeeld pagina 12-20.

9.3 Bulk DML

Niet alleen selecties kunnen in bulk gedaan worden, ook DML kan in bulk. Het commando hiervoor is de FORALL.

a) Lees de paragraaf "Running One DML Statement Multiple Times" op pagina 12-10 in de PL/SQL Reference Manual.

b) Gebruik de code uit 9.2b en in plaats van het tonen van de werknemersnamen, zet de namen in kleine letters met de LOWER-functie, met behulp van een FORALL-commando.

10. Dynamisch SQL

a) Lees hoofdstuk 7 uit de PL/SQL Reference Manual

b) Maak een database functie die een tabelnaam als invoerparameter heeft, en het aantal rijen in die tabel retourneert.