Low Brainers en No Brainers

Risico's bij code-aanpassingen verkleinen

De afgelopen decennia is een stuwmeer aan PL/SQL code ontstaan en dit groeit nog dagelijks. Veel business kritische processen zijn in PL/SQL geprogrammeerd en functioneren naar tevredenheid. De implementaties definiëren in zekere zin de processen. Zeker wanneer de systeemdocumentatie onvolledig is of slechts beschreven is op hoofdlijnen.

Wanneer de processen afhankelijk zijn van de implementatie is het cruciaal dat bij onderhoud de functionaliteit van de programmatuur niet onbedoeld wijzigt. Gebrek aan schaalbaarheid kan het noodzakelijk maken om onderhoud te plegen aan programmacode waarvan de functionaliteit op zich voldoet, bijvoorbeeld na een toename van klanten of gebruikers. Meer of betere hardware kan een oplossing bieden, maar is soms niet haalbaar of afdoende. Ook het huidige economische klimaat kan vragen dat systemen langer blijven draaien dan hun vooraf ingeschatte levensduur.

De nieuwste versies van PL/SQL bieden enkele manieren om performance en robuustheid van de code te verbeteren waarbij de risico's op onbedoelde functionaliteitswijzigingen zo klein mogelijk worden gehouden. Hoewel de winst bij de aanpassingen met het minste risico vaak ook het kleinst is, kan dit toch bijdragen aan de levensduurverlenging van een applicatie. Daarnaast kunnen dergelijke verbeteringen bij acute problemen tijd geven om meer risicovolle aanpassingen in te plannen.

In dit artikel wil ik enkele van deze mogelijkheden tot verbetering beschrijven zonder daarbij diep in te gaan op de details. De bedoeling is ook om inzicht te geven in welke mogelijkheden er zijn en stil te staan bij de balans die er bestaat tussen codeverbetering en het risico dat daarmee verbonden is.

Risico inschatting

Elke code-aanpassing introduceert een risico, afhankelijk van de grootte van de aanpassing. Slechts sommige configuratiewijzigingen kunnen worden gekenschetst als echte no-brainers.

Risico's kun je indelen naar gradatie:

- Geen: Code wordt niet gewijzigd
- Laag: Een enkele regel (Hint of PRAGMA) wordt toegevoegd ten behoeve van de compiler of interpreter
- Middel: Aanpassing van een keyword
- Hoog: Elke meer complexe aanpassing

De onderstaande mogelijkheden die Oracle biedt voor code-optimalisatie worden tegen deze meetlat aangelegd.

PLSQL_OPTIMIZE_LEVEL parameter

PLSQL_OPTIMIZE_LEVEL is een databaseparameter die aangeeft in hoeverre de code wordt geoptimaliseerd tijdens compilatie. Deze parameter is geïntroduceerd in Oracle 10g en kan de volgende waarden hebben:

0: Geen compiler optimalisatie
1: High level optimalisatie:
   - o verwijderen van constanten uit een loop
   - o Verwijderen van niet gebruikte toewijzingen of berekeningen

2: Default waarde, agressieve optimalisatie:
   - o Expressies die constante waardes opleveren worden uit een loop gehaald
   - o Als een conditie een constante waarde false oplevert wordt deze verwijderd
   - o Statische SQL cursor for loop wordt herschreven naar een BULK collect
   - o Ophogen van tellers is geoptimaliseerd. De x = x+1 instructie wordt niet meer in zijn geheel uitgevoerd maar vervangen door een enkele hiervoor geoptimaliseerde instructie.
   - o Een serie van concatinaties wordt samengevoegd in een statement.
   - o 'Inlining' van modules wanneer deze PRAGMA wordt gezet (zie verderop in dit artikel) .

3: Nieuw in Oracle 11g. Automatische inlining wanneer dit mogelijk is.

De default waarde is 2. Het zetten van deze parameter op de hoogst mogelijke waarde is een echte no-brainer. Oracle garandeert ongewijzigde functionaliteit, maar de compiler zal de code zoveel mogelijk optimaliseren. De daadwerkelijke winst is erg afhankelijk van de soort code maar kan factoren schelen. De volgende loop is met een setting van PLSQL_OPTIMIZE_SETTING van 3 vier maal zo snel als met een setting van 0.

```
ALTER SESSION SET PLSQL_OPTIMIZE_LEVEL = 0;

CREATE OR REPLACE FUNCTION f(c IN NUMBER) RETURN NUMBER IS
   z NUMBER := c;
   PROCEDURE p(x IN OUT NUMBER) IS BEGIN x := x + 1; END;
BEGIN
   FOR i IN 1 .. 10**7 LOOP
      p(z);
   END LOOP;
   RETURN z;
END;
/

SELECT f(1000) FROM dual;

   F(1000)
 ----------
  10001000
Elapsed: 00:00:03.56

ALTER SESSION SET PLSQL_OPTIMIZE_LEVEL = 3;

CREATE OR REPLACE ..
SELECT f(1000) FROM dual;

   F(1000)
 ----------
  10001000
Elapsed: 00:00:00.81
```

NATIVE compilation

Ook bij toepassen van deze optie hoeft geen code te worden aangepast. Native compilation houdt in dat de PL/SQL code wordt gecompileerd als C-routines op het operating systeem. Dit kan de code met een factor twee versnellen. De afhandeling van SQL wordt echter niet beïnvloed. Het gaat dus echt om de snelheid van de PL/SQL en deze optie heeft dus voornamelijk effect bij systemen die veel PL/SQL afhandelingen doen. Native compilation bestaat al sinds Oracle 9i. Tot nu toe was het hiervoor echter noodzakelijk om een C-compiler te installeren op de databaseservers. Dit is ook de reden dat native compilation tot nu toe weinig wordt gebruikt. Met Oracle 11g is de C-compiler niet meer nodig en ook de noodzaak voor het zetten van de *plsql_native_library_dir* parameter is verdwenen. Om Native compilation mogelijk te maken hoeft alleen nog maar een sessie parameter gezet te worden: *alter session set plsql_code_type = native;* Deze feature geeft vooral winst bij rekenintensieve toepassingen waarbij extra voordeel wordt behaald wanneer tevens gebruik gemaakt wordt van het datatype *SIMPLE_INTEGER*.

```
ALTER SESSION SET PLSQL_CODE_TYPE = 'NATIVE'

CREATE OR REPLACE ..

SELECT f(1000) FROM dual;

   F(1000)
 ----------
  10001000
Elapsed: 00:00:00.53
```

Subprogram inlining

Deze PRAGMA geeft aan dat de compiler een module zal 'inlinen' wat betekent dat tijdens de compilatie de inhoud van een module in zijn geheel gekopieerd wordt naar de plaats waar deze aangeroepen wordt. Tijdens de programmaverwerking vindt er geen aanroep van de module meer plaats. De code wordt direct uitgevoerd op de plaats waar het programma zich op dat moment bevindt.

```
    FOR i IN 1 .. 10**7 LOOP
      PRAGMA INLINE(p, 'YES');
       p(c);
    END LOOP;
```

Hierdoor neemt wel de grootte van de gecompileerde code en de compilatietijd toe, afhankelijk van de plsql_code_type parameter. Inlinen kan een versnelling opleveren van de code van 10-20%. Merk op dat wanneer de *PLSQL_OPTIMIZE_LEVEL* op 3 wordt gezet de inlining automatisch gebeurt.

PL/SQL Result Cache

PL/SQL result cache hint, volgens Steven Feuerstein de killer feature van Oracle 11G1, is een low-brainer, omdat dit wel degelijk een codewijziging betreft. Omdat deze optie een zeer gunstige invloed op performance kan hebben, dient toepassing ervan zeker te worden overwogen.
De result cache is een hint die wordt meegegeven aan een functie met als gevolg dat het resultaat van de functie wordt gecached. Dit houdt in dat indien deze functie al is uitgevoerd met dezelfde parameters en het resultaat nog in de cache is, de functie niet opnieuw hoeft te worden uitgevoerd, maar dat het resultaat uit de cache wordt gehaald. Wanneer de *RESULT_CACHE* hint wordt toegevoegd levert twee maal uitvoeren van *SELECT f(1000) FROM dual* het volgende resultaat op:

```
CREATE OR REPLACE FUNCTION f(c IN NUMBER) RETURN NUMBER RESULT_CACHE IS

          10001000
Elapsed: 00:00:00.53

          10001000
Elapsed: 00:00:00.01
```

Bij de tweede aanroep wordt de informatie volledig uit cache gehaald en de logica in de functie niet meer aangeroepen.

Ook eventuele queries in de functie worden niet meer uitgevoerd. Wat betreft database-afhankelijkheden is de cache volledig geautomatiseerd in die zin dat de cache automatisch wordt geschoond indien dit nodig is wegens het optreden van databasemutaties. In Oracle 11g1 moet voor een goede werking van de cache-opschoning nog worden aangegeven welke tabellen worden benaderd in de functie, maar in Oracle 11g2 is dat zelfs niet meer nodig en legt Oracle deze relatie automatisch. De result cache is een hele belangrijke nieuwe mogelijkheid in Oracle 11g die vooral winst oplevert wanneer modules vaak worden aangeroepen met dezelfde parameters. Wees erop bedacht dat de PL/SQL Result Cache in combinatie met Virtual Private Database toepassingen kan zorgen voor verkeerde resultaten. In VPD toepassingen is het mogelijk en wenselijk dat verschillende gebruikers een verschillend resultaat krijgen bij een gelijke query. Wanneer een functie resultaten ophaalt uit cache wordt mogelijk de verkeerde data teruggegeven, namelijk de gegevens behorend bij gebruiker die met zijn aanroep gezorgd heeft dat de functie met gegevens gecached werd.

NOCOPY hint

NOCOPY is een hint die ervoor zorgt dat bij de aanroep van submodules *IN OUT* en *OUT* parameters worden doorgegeven als referentie in plaats dat een lokale kopie wordt gemaakt van de parameter. Deze hint heeft vooral een gunstig effect in de gevallen dat er grote datastructuren worden doorgegeven zoals arrays. De *NOCOPY* hint heeft als nadeel dat indien een fout optreedt in de aangeroepen module de parameter niet wordt teruggegeven met de initiële waarde maar met de waarde zoals die op het moment van het optreden van de fout. Dit hoeft niet verkeerd te zijn maar is wel een afwijking van het standaardgedrag. Merk op dat de *NOCOPY* hint niet alleen de performance positief beïnvloed maar ook gunstig is voor het PGA gebruik.

```
DECLARE
   z CLOB := RPAD('x',10000,'x');
   PROCEDURE p(x IN OUT CLOB) IS BEGIN x := x||'y'; END;
BEGIN
   FOR i IN 1 .. 10**5 LOOP
      p(z);
   END LOOP;
END;
/
Elapsed: 00:00:19.23

DECLARE
   z CLOB := RPAD('x',10000,'x');
   PROCEDURE p(x IN OUT NOCOPY CLOB) IS BEGIN x := x||'y'; END;
BEGIN
   FOR i IN 1 .. 10**5 LOOP
      p(z);
   END LOOP;
END;
/
Elapsed: 00:00:05.29
```

Indien scalars worden meegegeven in plaats van data structuren of subprogram inlining optreedt wordt dit effect teniet gedaan.

Deterministic Hint

Deze hint bestaat al sinds Oracle 9i en geeft aan dat het resultaat van een functie volledig afhankelijk is van de parameters. Dit is een voorwaarde voor het gebruik van de functies in 'Function Based Indexes'. De resultaten van de functie worden echter ook gecached wanneer deze aangeroepen wordt in een SQL-statement dat meerdere rijen oplevert wat een performancewinst oplevert. Deze performancewinst kan in Oracle 11G ook worden gehaald door de *RESULT_CACHE* hint die minder beperkingen heeft.

```
CREATE OR REPLACE FUNCTION f(c IN NUMBER) RETURN NUMBER
DETERMINISTIC IS

SELECT f(1000) FROM obj WHERE rownum <= 3;

 F(1000)
----------
  10001000
  10001000
  10001000
Elapsed: 00:00:01.07
```

Datatype gebruik

Sinds Oracle 8i zijn er verschillende meer geoptimaliseerde datatypes geïntroduceerd zoals *PLS_INTEGER* en *SIMPLE_INTEGER*. Vooral deze laatste heeft een gunstig effect indien dit wordt gebruikt met Native compilation. Oracle raadt aan om constrainted datatypes zoals integer en positive te vermijden, *pls_integer* te gebruiken voor rekenkundige bewerkingen en *simple_integer* in combinatie met native compilation te overwegen in een Oracle 11g omgeving. Omdat hier altijd sprake is van code-aanpassingen is dit zeker geen no-brainer.

```
CREATE OR REPLACE FUNCTION f(c IN SIMPLE_INTEGER) RETURN NUMBER IS
   z SIMPLE_INTEGER := c;
   PROCEDURE p(x IN OUT SIMPLE_INTEGER) IS BEGIN x := x + 1; END;
BEGIN
   FOR i IN 1 .. 10**7 LOOP
      p(z);
   END LOOP;
   RETURN z;
END;

SELECT f(1000) FROM dual;

   F(1000)
 ----------
  10001000
Elapsed: 00:00:00.07
```

Het *NUMBER* datatype veranderen van *NUMBER* naar *SIMPLE_INTEGER* levert in bovenstaand voorbeeld een versnelling op met factor 5.

Tabel 1: Doorlooptijd van de verschillende scenario's

Optie	Hint of parameter	Instelling	Tijd in Seconden
Oracle 9i instelling	PLSQL_OPTIMIZE_LEVEL	0	3.56
Oracle 10g behoudend	PLSQL_OPTIMIZE_LEVEL	1	2.43
Oracle 10g default instelling	PLSQL_OPTIMIZE_LEVEL	2	1.98
Inclusief automatische inlining	PLSQL_OPTIMIZE_LEVEL	3	0.81
Native compilation	PLSQL_CODE_TYPE	NATIVE	0.53
Parameter by reference hint	NOCOPY		0.53
Datatype gebruik	SIMPLE_INTEGER		0.07
Result cache hint	RESULT_CACHE		0.01

Tabel 2: Risico op introductie van fouten per optimalisatie methode.

Optimalisatie methode	Aanpassing	Risico	Verwacht positief effect
PLSQL_OPTIMIZE_LEVEL	Database parameter	Geen	Klein – Middel
Native compilation	Sessie parameter	Geen	Klein – Middel
intraunit inlining impliciet doormiddel van parameter setting	Database parameter	Geen	Klein – Middel
intraunit inlining expliciet	Compiler PRAGMA	Laag	Klein – Middel
PL/SQL result cache	PL/SQL Hint	Laag	Groot
NOCOPY hint	PL/SQL Hint	Laag	Klein – Middel
DETERMINISTIC hint	PL/SQL Hint	Laag	Klein – Middel
Datatype gebruik	Code wijziging	Middel	Klein – Middel
BULK collects en FORALL	Code wijziging & herstructurering	Hoog	Groot

Code refactoring: Bulk collects en FORALL clause.

Het aanpassen van traditionele Query afhandeling naar *BULK COLLECTS* en traditionele DML in loops naar *FORALL* loops behoort tot de meer complexe en daardoor risicovollere aanpassingen. Toch noemt Steven Feuerstein dit samen met de result cache de belangrijkste optimalisatiemogelijkheid. Zeker in praktijksituaties waarbij traagheid het gevolg is van database-interactie. Wel dient in de gaten te worden gehouden dat het PGA geheugengebruik toeneemt. Bulk collects zouden altijd gelimiteerd moeten worden met de *LIMIT* clause. Daarnaast vereist BULK processing een andere foutafhandeling dan traditionele SQL afhandeling. Ook moet men bedacht zijn op functionaliteit wijzigingen. Bijvoorbeeld de statement triggers vuren slechts eenmaal per *FORALL* statement, waar deze bij een traditionele *FOR* loop voor elk update statement afgaan.

PL/SQL compiler warnings

Sinds Oracle versie 10 heeft Oracle compiler warnings geïntroduceerd. Door middel van het zetten van de *PLSQL_WARNINGS* sessieparameter worden bij compilatie warnings conform errors gegeven met het verschil dat wanneer slechts warnings optreden de code wel wordt gecompileerd. Deze warnings attenderen op zaken als onbereikbare code, datatype conversieproblemen, verkeerd gebruik van aliassen. In het geval van code refactoring is het een goed idee om deze parameter aan te zetten. Omdat de compiler warning optie nog in ontwikkeling is, kan het zelfs handig zijn om indien mogelijk de code te compileren tegen een hogere versie van Oracle dan waar de code in productie op draait om inzicht te krijgen in waar eventuele verbeteringen mogelijk zijn.

Conclusie

Tabel 1 geeft een overzicht van het effect van de verschillende no - and low brainer opties.

Oracle reikt enkele hulpmiddelen aan om zonder of met minimale code wijzigingen de schaalbaarheid en robuustheid van PL/SQL programmatuur te verbeteren. Tabel2 geeft een overzicht van de mogelijkheden en het risico op onbedoelde functionaliteitwijzigingen. Het effect van de wijziging is afhankelijk van de implementatie details. Zo zal gebruik van 'result cache' alleen voordeel bieden wanneer functies vaker met dezelfde parameters worden aangeroepen. Omdat het echter niet zo is dat de meer risicovolle wijzigingen ook de meeste winst opleveren is het raadzaam om alle mogelijkheden te overwegen in gevallen waarbij technisch onderhoud van een applicatie nodig is.

Gerard van der Elst is IT Consultant bij Logica.