Oracle 12c: Neuerungen in PL/SQL

Roman Pyro
Herrmann & Lenz Services GmbH
Burscheid

Schlüsselworte
Neuerungen PL/SQL 12c, Exception Handling, Rechteverwaltung, Kontextwechsel SQL- / PL/SQL-Engine

Einleitung
Im Rahmen der Veröffentlichung der neuen Oracle 12c Datenbankversion wurde auch PL/SQL um diverse neue Funktionalitäten erweitert. Ein Teil der Erweiterungen betrifft die folgenden Themen:
Galt die Rechteverwaltung über Rollen im PL/SQL Umfeld bisher als wenig praktikabel, wurde hier die Möglichkeit geschaffen, Rollen direkt an Funktionen zuzuweisen.
Die Möglichkeit Tabellenspalten unsichtbar zu machen mag auf den ersten Blick wie eine Möglichkeit erscheinen, bestimmte Informationen vor unbefugten Zugriffen zu schützen, ist jedoch primär für Tabellenerweiterungen sinnvoll.
Ein elementarer Bestandteil jedes PL/SQL Programms ist das Exceptionhandling. Hier wurden die Möglichkeiten der Analyse über das UTL_CALL_STACK Package erweitert.
Wurde in der Vergangenheit eine PL/SQL Funktion innerhalb eines SELECT aufgerufen, erzeugte dies einen Kontextwechsel zwischen SQL-Engine und PL/SQL-Engine. Mit Oracle 12c wurde die Möglichkeit geschaffen, eine PL/SQL Funktion innerhalb einer WITH Klausel zu definieren bei deren Aufruf kein Kontextwechsel stattfindet.

Rechtevergabe auf Funktionen
Bis einschließlich Oracle 11g gab es zwei Möglichkeiten der Rechteverwaltung für Funktionsaufrufe.
Bei AUTHID CURRENT_USER (Invoker Rights) wird eine Funktion mit den Rechten des aufrufenden Users ausgeführt. Je nach Architektur muss man hier den Benutzern sehr viel mehr Rechte einräumen, als gewünscht.
Standardmäßig wird AUTHID DEFINER (Definer Rights) genutzt. Hier benötigt der aufrufende Benutzer Execute Rechte auf die Funktion / Prozedur / Package. Die Zugriffsrechte auf Objekte welche innerhalb der Funktion aufgerufen werden, müssen dem Besitzer der Funktion gegeben sein. Zu beachten ist hier, dass über Rollen vergebene Rechte nicht gewertet werden. Ausschließlich direkt vergebene Rechte zählen.
Seit Oracle 12c ist es möglich Rechte direkt an ein Funktion zu vergeben. Dazu wird die Funktion mit Invoker Rights definiert und anschließend kann ein Grant vergeben werden.

Die folgende Funktion zählt die im letzten Monat neu erstellten Benutzer auf Basis der Tabelle DBA_USERS und gibt diese als Number Wert zurück. Da die Funktion mit Invoker Rights definiert ist, muss der Aufrufende Benutzer die Select Rechte auf die DBA_USERS Tabelle bekommen.

```
CREATE OR REPLACE FUNCTION rpy.get_new_users
   RETURN NUMBER
   AUTHID CURRENT_USER
IS
   l_count  NUMBER;
BEGIN
   SELECT COUNT(*)
     INTO l_count
     FROM dba_users usr
    WHERE USR.CREATED BETWEEN ADD_MONTHS(TRUNC(SYSDATE, 'MM'), -1)
                          AND LAST_DAY(ADD_MONTHS(TRUNC(SYSDATE, 'MM'), -1));

   RETURN l_count;
END get_new_users;
/
```

Um einen ungefilterten Zugang zu der Tabelle DBA_USERS zu vermeiden, kann nun ein grant an die Funktion übergeben werden.

```
GRANT SELECT_CATALOG_ROLE TO rpy;
GRANT SELECT_CATALOG_ROLE TO function rpy.get_new_users;
```

Zu beachten ist hier das eine Funktion nur ein Recht zugesprochen bekommen kann, welches auch der Besitzer der Funktion erhalten hat. Bei einem Funktionsaufruf durch weitere Nutzer wird die Funktion nun mit den Rechten des aufrufenden Users (Invoker Rights) ausgeführt, sowie der SELECT_CATALOG_ROLE. Der aufrufende Benutzer hat jedoch weiterhin nicht die Möglichkeit direkt die Tabelle dba_users zu selektieren.

Neue Möglichkeiten im Exceptionhandling

Mit Oracle 12c wurde PL/SQL um das Package UTL_CALL_STACK erweitert. Dieses ermöglicht es, auf den Callstack zuzugreifen. Im Gegensatz zu dem bereits bestehenden `dbms_utility.format_call_stack` Aufruf kann auf die Einzelinformationen jedoch explizit zugegriffen werden. Möchte man den Callstack im Exceptionhandling auswerten und eventuell in eine Logging Tabelle schreiben, ist es nun nicht mehr notwendig die Informationen durch einen Parser laufen zu lassen.

Ausgabe dbms_utility.format_call_stack:

```
BEGIN
[..]
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.Put_Line(DBMS_UTILITY.Format_Call_Stack);
END ;
/

----- PL/SQL Call Stack -----
  object      line  object
  handle    number  name
000007FF5C134948        13  function RPY.GET_RESULT
000007FF5BFB5C00         6  function RPY.GET_DAY_OF_MONTH
000007FF5BFB6220         4  anonymous block
```

Ausgabe mittels utl_call_stack:

```
BEGIN
[..]
EXCEPTION
   WHEN OTHERS THEN
      FOR i IN REVERSE 1 .. utl_call_stack.dynamic_depth()
      LOOP
         DBMS_OUTPUT.Put_Line(
               'Zeile: '
            || UTL_Call_Stack.Unit_Line(i)
            || ' Program: '
            || UTL_Call_Stack.Concatenate_Subprogram(
                  UTL_Call_Stack.Subprogram(i)));
      END LOOP;
END ;
/

Zeile: 4 Program: __anonymous_block
Zeile: 6 Program: GET_DAY_OF_MONTH
Zeile: 25 Program: GET_RESULT
```

Wie zu sehen ist, werden im utl_call_stack Package die Informationen explizit abgerufen. Vor allem für Funktionen welche durch diverse externe Programme aufgerufen werden eignet sich dies sehr gut um Informationen im Fehlerfall in eine Logging-Tabelle zu schreiben.

Schnellerer PL/SQL Code innerhalb einer WITH-Klausel
Seit Oracle 12c ist es möglich PL/SQL Funktionen und Prozeduren direkt in einer SQL WITH-Klausel zu definieren ohne diese in der Datenbank kompilieren zu müssen.

Für SQL Statements kann dies interessant sein, da PL/SQL Code innerhalb einer WITH-Klausel eine schnellere Laufzeit vorweist als Aufrufe von kompilierten Funktionen.

Um dies nachzuvollziehen kann ein Select Statement einmal mit einer WITH-klausel ausgeführt werden und einmal mit einem klassischen Funktionsaufruf. Eine Zeitmessung kann über DBMS_UTILITY.get_time / DBMS_UTILITY.get_cpu_time durchgeführt werden.

Für den Test wurde eine Testtabelle t_with_perf angelegt. Diese enthält eine Spalte (id number) und 100.000 Datensätze. Das Select wird für jeden Datensatz die ID an eine Funktion übergeben welcher dann wieder zurückgegeben wird.

Select mit klassischem Aufruf:

```
SELECT get_value(id)
  FROM t_with_perf;
```

Select mit WITH-Klausel:

```
WITH
    FUNCTION get_value_with(p_id IN NUMBER) RETURN NUMBER IS
             BEGIN
                RETURN p_id;
             END;
SELECT get_value_with(id)
  FROM   t_with_perf
```

In allen Testläufen benötigte der Aufruf mit einer WITH-Klausel etwa 2/5 der Zeit eines klassischen Funktionsaufrufs.

Unsichtbare Tabellenspalten
Ein weiteres neues Feature seit Oracle 12c sind Invisible Columns. Entgegen der Bezeichnung sind Tabellenspalten welche entsprechend deklariert sind jedoch nur in bestimmten Szenarien unsichtbar.

```
CREATE TABLE test_invisible
(
   visible_column    VARCHAR2(10),
   invisible_column  VARCHAR2(10) INVISIBLE
);
```

Möchte man die Tabelle in einem Programm nutzen und prüfen welche Spalten die Tabelle hat z.B. mittels dem folgenden select kann man sehen dass die Spalte durchaus angegeben wird.

```
SELECT column_name,
       data_type,
       data_length,
       hidden_column
  FROM all_tab_cols
 WHERE table_name = 'TEST_INVISIBLE';
```

```
COLUMN_NAME       DATA_TYPE   DATA_LENGTH   HIDDEN_COLUMN
----------        ---------   -----------   -------------
VISIBLE_COLUMN    VARCHAR2    10            NO
INVISIBLE_COLUMN  VARCHAR2    10            YES

2 rows selected.
```

Invisible Columns sind ganz klar kein Mittel um Daten vor unbefugten zu verstecken und ergänzen auch kein Rollen- / Berechtigungs-Konzept.

Interessant im PL/SQL Umfeld sind Invisible Columns jedoch, sobald ein Datenmodell erweitert werden soll und nicht ausgeschlossen werden kann, dass es `select * from …` Zugriffe auf die Tabelle gibt oder `insert into <TABELLE> values(…);`

```
BEGIN
    INSERT INTO test_invisible
          VALUES ('test');

    COMMIT;
EXCEPTION
    WHEN OTHERS
    THEN
        ROLLBACK;
        RAISE;
END;
/
```

Normalerweise würde der Code nach Zufügen einer Spalte nicht mehr funktionieren. Da die Spalte mit dem Invisible-Attribut definiert wurde, wird der PL/SQL Block jedoch weiter Funktionieren.

Immer wieder findet man select * from … Statements in PL/SQL Code. Derartiger Code – wie im folgenden Beispiel zu sehen – würde nicht mehr funktionieren, wäre eine zugefügte Spalte nicht Invisible.

```
DECLARE
   TYPE lr_test IS RECORD(visible VARCHAR2(10));

   TYPE lt_test IS TABLE OF lr_test
      INDEX BY VARCHAR2(10);
   l_test  lt_test;
BEGIN
   FOR i IN (SELECT * FROM test_invisible)
   LOOP
      l_test(i.visible_column)  := i;
   END LOOP;
EXCEPTION
   WHEN OTHERS THEN
      ROLLBACK;
      RAISE;
END;
/
```

Würde die Spalte invisible_column mit ausgegeben, käme der folgende Fehler.

```
Error at line 1
ORA-06550: Zeile 11, Spalte 36:
PLS-00382: Dieser Ausdruck hat den falschen Typ
ORA-06550: Zeile 11, Spalte 7:
PL/SQL: Statement ignored
```

Weitere Themen
Weitere Neuerungen sind unter anderem:

- Fetch-First-Clause zum begrenzen der Ergebnismenge
- Accessible-By-Clause zum Erstellen von White-Lists

Kontaktadresse:
Roman Pyro
Herrmann & Lenz Services GmbH
Höhestr. 37
51399 Burscheid

Telefon: +49 (0) 2174-6712-294
Fax: +49 (0) 2174-6712-22
E-Mail roman.pyro@hl-servies.de
Internet: hl-services.de