Herrmann & Lenz
Services

Oracle 12c: Neuerungen in PL/SQL

Roman Pyro

DOAG 2014 Konferenz

Herrmann & Lenz Services GmbH
Herrmann & Lenz Solutions GmbH

- Erfolgreich seit 1996 am Markt
- Firmensitz: Burscheid (bei Leverkusen)
- Beratung, Schulung und Betrieb/Fernwartung rund um das Thema Oracle Datenbanken
- Schwerpunktthemen: Hochverfügbarkeit, Tuning, Migrationen und Troubleshooting
- Herrmann & Lenz Solutions GmbH
  - Produkt: Monitoring Module
  - Stand auf Ebene 2

Übersicht

- Code Based Access Control (CBAC)
- Accessible-By-Clause
- Call-Stack Zugriff über UTL_CALL_STACK
- PL/SQL in Subquery Factoring Clause
- Invisible Columns
- Row Limiting Clause

Code Based Access Control (CBAC)

Bisherige Möglichkeiten

Invoker‘s Rights	Definer‘s Rights (Default)
Läuft mit Rechten des Aufrufenden	Läuft mit Rechten des Besitzers
Rollen werden bei Ausführung berücksichtigt	Rollen werden nicht berücksichtigt
Muss in der Funktionsdeklaration definiert werden	Standardeinstellung für alle Funktionen, Prozeduren und Packages
CURRENT_USER / CURRENT_SCHEMA werden bei Ausführung nicht verändert	CURRENT_USER / CURRENT_SCHEMA werden auf Besitzer der Funktion gesetzt

Neu Hinzugekommen

- Rechtevergabe an Funktionen

- Voraussetzungen:
  - Definer muss entsprechendes Recht erhalten
  - Execute Recht für Invoker

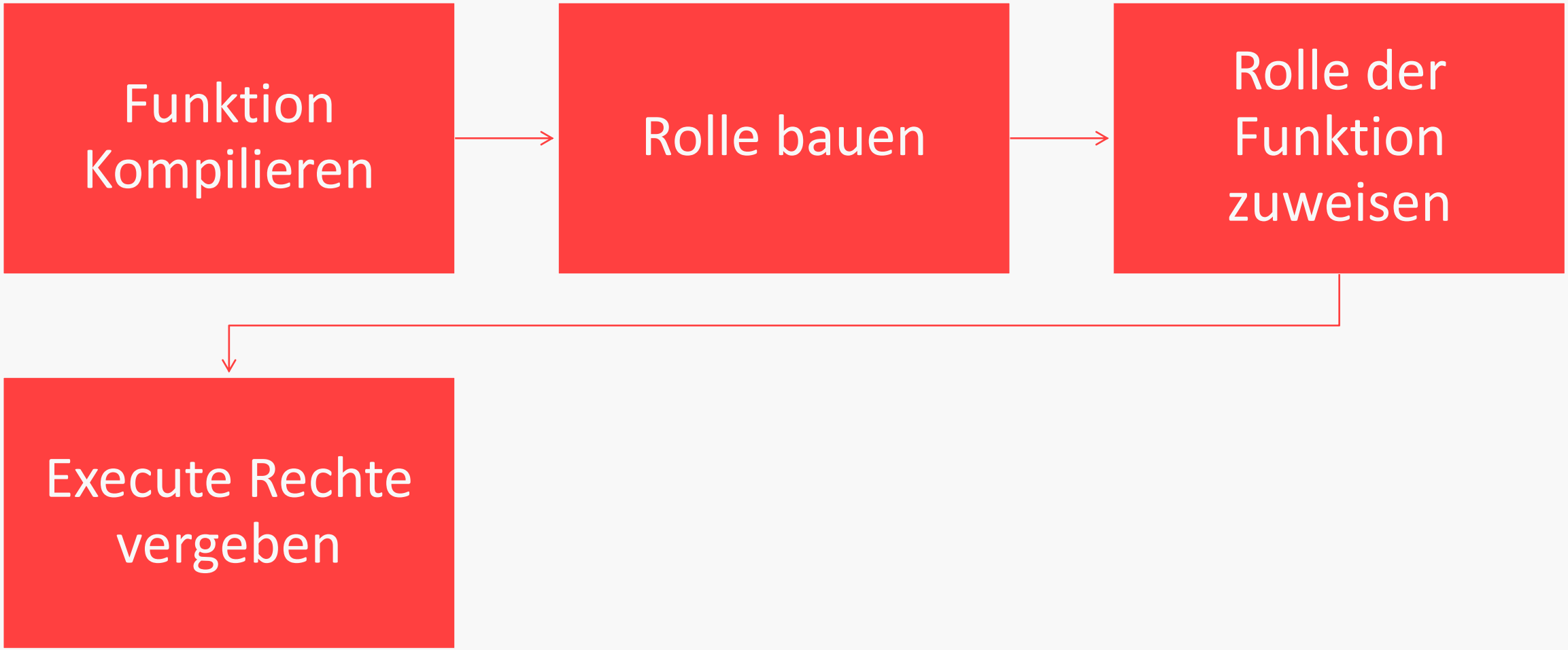
Beispiel

```
CREATE OR REPLACE PROCEDURE
cbac_owner.call_test
   AUTHID CURRENT_USER
AS
   l_count   NUMBER;
BEGIN
   SELECT COUNT(*)
     INTO l_count
     FROM cbac_owner.test;
END call_test;
```

```
--Rolle an Prozedur vergeben
CREATE ROLE cbac_role;
GRANT SELECT ON test TO cbac_role;
GRANT cbac_role TO PROCEDURE
call_test;
```

```
--Execute Recht vergeben
GRANT EXECUTE ON call_test TO
cbac_caller;
```

Zusammenfassung



Rollen beim Kompilieren

- Invoker‘s Rights & Definer‘s Rights verhalten sich beim Kompilieren wie Definer‘s Rights

- Kompilieren mit Fremdschema-Zugriff:
  - Dynamisches SQL
  - Rechte direkt an Definer vergeben

Vergissmeinnicht

- Funktions-Grants werden bei Funktions-Änderung vergessen

- Funktions-Grants neu setzen bei:
  - Änderung an Prozedur
  - Änderung an Funktion
  - Änderung an Package Header

Accessible-By-Clause

Was ist eine Accessible-By-Clause?

- Funktionen können Whitelist definieren
- Aufruf nur durch Funktionen auf Whitelist
- Gilt auch für Debug Session / Anonymer Block

Beispiel

```
--Prozedur mit Whitelist
CREATE OR REPLACE PROCEDURE
prc_access_by
   ACCESSIBLE BY(rpy.call_ok)
AS
BEGIN
DBMS_OUTPUT.put_line(SYSDATE);
END;
/
GRANT EXECUTE ON prc_access_by
TO rpy;
```

```
--Wird Kompiliert
CREATE OR REPLACE PROCEDURE
call_ok AS
BEGIN
   test.prc_access_by;
END;
/
--Fehler: PLS-00904
CREATE OR REPLACE PROCEDURE
call_fail AS
BEGIN
   test.prc_access_by;
END;
/
```

Call Stack Zugriff über UTL_CALL_STACK

UTL_CALL_STACK

- Bietet Zugriff auf Call Stack / Error Stack
- Mehr Möglichkeiten als DBMS_UTILITY
- Strukturierter Zugriff auf Stack Informationen

Beispiel

```
[..]
FOR i IN REVERSE 1 .. utl_call_stack.dynamic_depth()
LOOP
   DBMS_OUTPUT.put_line(
     'Ebene: ' || i ||
     '  Zeile: ' || utl_call_stack.unit_line(i)||
     '  Programm: ' ||
           utl_call_stack.concatenate_subprogram(
               utl_call_stack.subprogram(i)));
END LOOP;
[..]
```

Beispiel

UTL_CALL_STACK

Ebene: 3 Zeile: 4 Programm: .__anonymous_block
Ebene: 2 Zeile: 5 Programm: RPY.GET_DAY_OF_MONTH
Ebene: 1 Zeile: 12 Programm: RPY.GET_RESULT

DBMS_UTILITY.FORMAT_CALL_STACK

```
----- PL/SQL Call Stack -----
  object      line  object
  handle    number  name
000007FF5A75E5A0        12  function RPY.GET_RESULT
000007FF5A6C4888         5  function RPY.GET_DAY_OF_MONTH
000007FF5A6BDC58         4  anonymous block
```

PL/SQL in Subquery Factoring Clause

Was ist neu?

- PL/SQL Funktionen in With-Clause deklarieren
- Nur zur Laufzeit gültig
- Kann Performance-Gewinn ermöglichen

Beispiel

```
--WITH-Clause
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;
```

```
--Klassisch
CREATE OR REPLACE FUNCTION
get_value(p_value IN NUMBER)
RETURN NUMBER IS
BEGIN
   RETURN p_value;
END;
/

SELECT get_value(id)
  FROM t_with_perf;
```

Performance Messung

Aufruf	Anzahl Durchläufe	Total CPU	Total Elapsed
Funktionsaufruf	500	514s	514s
WITH-Clause	500	185s	185s

WITH-Clause benötigt etwa 35% der Zeit eines Funktionsaufrufs.

Invisible Columns

Wirklich Unsichtbar?

- Werden bei SELECT * FROM / INSERT INTO x VALUES() nicht genutzt

- Können direkt angesprochen werden

Wozu?

- SELECT * FROM Statements ändern Verhalten nicht
- Nicht als Zugriffsbeschränkung zu verwenden

Beispiel

```
DECLARE
   TYPE lr_visible IS RECORD(column_visible
      test_invisible.visible_column%TYPE);
   l_record      lr_visible;
BEGIN
   FOR i IN (SELECT * FROM test_invisible)
   LOOP
      l_record := i;
   END LOOP;
END;
/
```

Row Limiting Clause

Wie und wo einsetzbar?

```
--Die ersten 5 Datensätze
  SELECT tff.PK_VALUE
    FROM test_fetch_first tff
ORDER BY tff.pk_value ASC
FETCH FIRST 5 ROWS ONLY;

--Weitere 5 Datensätze
--beginnend bei offset + 1
  SELECT tff.PK_VALUE
    FROM test_fetch_first tff
ORDER BY tff.pk_value ASC
  OFFSET 5 ROWS
FETCH NEXT 5 ROWS ONLY;
```

- Kann anstelle ROWNUM genutzt werden
- Übersichtlichere Syntax

Fragen & Kontakt

- roman.pyro@hl-services.de
- http://www.hl-services.de
- Hier in der Ausstellung Stand 212: Ebene 2 (gelb) direkt gegenüber der Rolltreppe
- Besuchen Sie auch unseren Vortrag:
  - Datenbankkonsolidierung: Multitenant oder nicht? Dierk Lenz, 18.11. 11:00 Uhr, Saal Tokio
  - Platzersparnis in einer 10-TB-Datenbank durch Reorganisation, Susanne Jahr 19.11. 11:00 Uhr, Saal Tokio