Tipps & Tricks: DBMS_UTILITY

Bereich:	PL/SQL	Erstellung:	03/2002 HA
Versionsinfo:	8.1, 9.0, 9.2, 10.2, 11.1	Letzte Überarbeitung:	06/2009 HA

Verwendung von DBMS_UTILITY

Dieses Package bietet eine Reihe von Prozeduren und Funktionen für die verschiedensten Fragestellungen. Ein paar davon sollen hier vorgestellt werden.

GET_TIME

Diese Funktion ist die gebräuchlichste aus dem Package. Sie wird in der Regel dazu verwendet, die Dauer der Ausführung eines oder mehrerer Befehle zu messen. GET_TIME gibt zwar die Zeit in Hundertstel Sekunden zurück, kann aber nicht dazu verwendet werden, ein Datum in eine Zahl umzuwandeln, weil der Anfangspunkt nicht definiert ist.

Beispiel:

```
DECLARE
   v_dummy     NUMBER;
   v_dummy2     NUMBER;
   -- sonstige Deklarationen
BEGIN
   v_dummy := DBMS_UTILITY.GET_TIME;
   -- z. B. Prozeduraufruf
   v_dummy2 := DBMS_UTILITY.GET_TIME;
   DBMS_OUTPUT.PUT_LINE('Verstrichene Zeit (in Hundertstel Sekunden): '
       ||(v_dummy2 - v_dummy));
END;
/
```

Anmerkung:
Seit Version 10g gibt es zusätzlich die Funktion GET_CPU_TIME. Während mit GET_TIME die gesamte Dauer der Ausführung gemessen werden kann, bekommt man mit GET_CPU_TIME heraus, wie viel Zeit davon durch die CPU verbraucht wurde.

IS_PARALLEL_SERVER / IS_CLUSTER_DATABASE, CURRENT_INSTANCE DB_VERSION, PORT_STRING

Diese Funktionen liefern Informationen über die Datenbank und das Betriebssystem. IS_PARALLEL_SERVER (bis einschließlich Version 9.0.x) bzw. IS_CLUSTER_DATABASE (ab Version 9.2) geben einen Boole'schen Wert zurück, der angibt, ob die Datenbank im Parallel Server-Modus (bzw. Cluster-Modus) läuft. Seit Version 9.2.x gibt es die Funktion IS_PARALLEL_SERVER nicht mehr; ein Skript, das gleichzeitig auf älteren und neueren Versionen läuft, ist daher nur über dynamisches SQL zu realisieren. CURRENT_INSTANCE liefert seit Version 8i die Nummer der aktuell verbundenen Instanz.
DB_VERSION liefert Informationen über die Version und die Kompatibilitätseinstellung der Datenbank.
PORT_STRING liefert Informationen über das Betriebssystem des Servers und die TWO TASK Protokoll-Version der Datenbank.

Beispiel:

```
DECLARE
   v_version    VARCHAR2(2001);
   v_comp       VARCHAR2(2001);
   v_os        VARCHAR2(2001);
   v_instance   NUMBER;

BEGIN
   DBMS_UTILITY.DB_VERSION(v_version, v_comp);
   DBMS_OUTPUT.PUT_LINE('Version: ' || v_version);
   DBMS_OUTPUT.PUT_LINE('Kompatibilität: ' || v_comp);
   v_os := DBMS_UTILITY.PORT_STRING;
   DBMS_OUTPUT.PUT_LINE('Betriebssystem und Protokoll: ' || v_os);
   --IF DBMS_UTILITY.IS_PARALLEL_SERVER THEN -- nur bis Version 9.0.x
   IF DBMS_UTILITY.IS_CLUSTER_DATABASE THEN  -- seit Version 9.2
      DBMS_OUTPUT.PUT_LINE('Parallel Server Modus');
      v_instance := DBMS_UTILITY.CURRENT_INSTANCE;
      DBMS_OUTPUT.PUT_LINE('aktuelle Instanz: ' || v_instance);
   ELSE
      DBMS_OUTPUT.PUT_LINE('Kein Parallel Server Modus');
   END IF;
END;
```

Ausgabe bei einer Sun Ultra60:

```
    Version: 9.0.1.0.0

    Kompatibilität: 9.0.0

    Betriebssystem und Protokoll: SVR4-be-8.1.0

    Kein Parallel Server Modus
```

Ausgabe bei Windows-Systemen (NT, 2000, 2003 Server und 2008 Server):

```
   Version: 8.1.7.0.0

   Kompatibilität: 8.1.0

   Betriebssystem und Protokoll: IBMPC/WIN_NT-8.1.0

   Kein Parallel Server Modus
```

GET_PARAMETER_VALUE

Diese Funktion liefert den Wert und den Typ eines Init.ora-Parameters zurück; der Name muss dabei in richtiger Schreibweise (Kleinbuchstaben) übergeben werden. Sie steht auch Usern zur Verfügung, die kein SELECT-Recht auf V$PARAMETER haben(!).

Falls es sich um einen String handelt, wird 1 zurückgegeben, für Integer oder Boolean wird 0 zurückgegeben. Bei Boole'schen Werten wird für FALSE 0 und für TRUE 1 als Wert geliefert. Bei Strings kann zusätzlich die Länge des Strings mit ausgelesen werden.

Beispiel:

```
CREATE OR REPLACE FUNCTION GET_VALUE (p_name IN VARCHAR2)
RETURN  VARCHAR2
IS
   v_type       BINARY_INTEGER;
   v_value_int  BINARY_INTEGER;
   v_value      VARCHAR2(2001);

BEGIN
   v_type := DBMS_UTILITY.GET_PARAMETER_VALUE
                        (LOWER(p_name), v_value_int, v_value);
   IF v_type = 0 THEN   --Integer oder Boolean
      RETURN TO_CHAR(v_value_int);
   ELSIF v_type =1 THEN --String oder File
      --DBMS_OUTPUT.PUT_LINE('Länge des Wertes: '||v_value_int);
      RETURN (v_value);
   END IF;
EXCEPTION
   WHEN OTHERS THEN
      RETURN '';
END;
/

EXEC DBMS_OUTPUT.PUT_LINE(GET_VALUE('background_dump_dest'))
EXEC DBMS_OUTPUT.PUT_LINE(GET_VALUE('processes'))
EXEC DBMS_OUTPUT.PUT_LINE(GET_VALUE('global_names'))
```

COMMA_TO_TABLE, TABLE_TO_COMMA

Diese Prozeduren bieten die Möglichkeit, Namen vom Objekten von einer Komma separierten Liste in eine PL/SQL-Tabelle zu überführen und umgekehrt. Dabei wird nicht überprüft, ob das Objekt existiert, aber es wird überprüft, ob die Namenskonventionen eingehalten wurden. Der letzte Wert des Arrays wird bei COMMA_TO_TABLE mit NULL belegt.

Beispiel:

```
DECLARE
   v_tab       DBMS_UTILITY.UNCL_ARRAY;
   v_list      VARCHAR2(32000);
   v_list_new  VARCHAR2(32000);
   v_count     BINARY_INTEGER;
   v_count_new BINARY_INTEGER;

BEGIN
   v_list := 'asd2,xyz,"2emp",dept,'||
      'ein_maximal_30_Zeichen_name.und_nochmal_maximal_30_zeichen';
   DBMS_UTILITY.COMMA_TO_TABLE(v_list, v_count, v_tab);
```

```
   DBMS_OUTPUT.PUT_LINE('Ursprüngliche Anzahl der Einträge: '
                                                || v_count);
   FOR i IN v_tab.FIRST.. v_tab.LAST -1 LOOP --letzter Wert: NULL
      DBMS_OUTPUT.PUT_LINE(v_tab(i));
   END LOOP;
   v_tab(v_tab.LAST) := 'neuer_eintrag';
   v_tab(v_tab.LAST+1) := NULL;
   DBMS_UTILITY.TABLE_TO_COMMA (v_tab, v_count_new, v_list_new );
   DBMS_OUTPUT.PUT_LINE('Spätere Anzahl der Einträge: '
                                        || v_count_new);
   DBMS_OUTPUT.PUT_LINE('Liste: ' || v_list_new);
END;
/
```