Dynamisches SQL

Dynamisches SQL

Dynamisch heisst, dass die Struktur der SQL-Anweisung erst zur Laufzeit feststeht.

- Wird als Zeichenkette durch die Applikation aufgebaut und anschließend dynamisch als Anweisung ausgeführt.
- Damit können DDL-, DCL- und Sitzungskontroll-Anweisungen aus PL/SQL heraus ausgeführt werden.
- Abhängigkeiten und Syntax-Fehler können jetzt aber erst zur Laufzeit geprüft werden.
- Die Umsetzung kann mit NDS (Native Dynamic SQL) oder dem Paket DBMS_SQL erfolgen.

Oracle unterscheidet folgende Methoden (Level) bei der Ausführung von dynamischen SQL

1. SQL-Anweisungen ohne Abfrage und ohne Verwendung von BIND-Variablen: NDS oder DBMS_SQL
2. SQL-Anweisungen ohne Abfrage und mit Verwendung von BIND-Variablen: NDS oder DBMS_SQL.
3. Abfrage mit bekannter Anzahl von Elementen in der SELECT-Liste und von BIND-Variablen: NDS oder DBMS_SQL
4. Abfrage mit unbekannter Anzahl von Elementen in der SELECT-Liste oder von BIND-Variablen: nur DBMS_SQL.

Native Dynamic SQL (NDS)

- Direkte Unterstützung für dynamisches SQL in PL/SQL.
- Erlaubt die Ausführung von SQL-Anweisungen, deren Struktur erst zur Laufzeit feststeht.
- Ab Oracle 11g werden Anweisungen mit mehr als 32KB Länge via CLOB unterstützt.
- Parametrisierung ist mittels BIND-Variablen möglich.
- Falls es eine SELECT-Anweisung ist, die mehrere Zeilen zurückliefert, kann die Ergebnismenge auf 2 Arten verarbeitet werden:
  - `EXECUTE IMMEDIATE` -Anweisung mit `BULK COLLECT INTO` -Klausel
  - Via Cursor mit `OPEN-FOR`, `FETCH`, und `CLOSE`-Anweisungen

Verwenden von `EXECUTE IMMEDIATE`

Erlaubt die Ausführung von SQL-Anweisungen oder anonymen PL/SQL-Blöcken:

```
EXECUTE IMMEDIATE dynamic_string
 [INTO {define_variable
     [, define_variable] ... | record}]
 [USING [IN|OUT|IN OUT] bind_argument
     [, [IN|OUT|IN OUT] bind_argument] ... ];
```

- `INTO` : gibt die Variablen bzw. Records für die Ergebnisse bei Single-Row-Abfragen an.
- `USING` : gibt die benutzten BIND-Variablen an. Standard-Prametermodus ist `IN`.
- Erlaubte Typen für die BIND-Variablen sind Zahlen und Zeichenketten, aber keine literalen Wahrheitswerte (`TRUE, FALSE, NULL`)

Beispiel Methode 1: NDS mit DDL-Anweisung

```
-- Create a table using dynamic SQL

CREATE OR REPLACE PROCEDURE create_table(
  p_table_name VARCHAR2, p_col_specs  VARCHAR2) IS
BEGIN
  EXECUTE IMMEDIATE 'CREATE TABLE ' || p_table_name ||
                    ' (' || p_col_specs || ')';
END;
/
```

```
-- Call the procedure

BEGIN
  create_table('EMPLOYEE_NAMES',
      'id NUMBER(4) PRIMARY KEY, name VARCHAR2(40)');
END;
/
```

Beispiel Methode 1: NDS mit DML-Anweisung

```
-- Delete rows from any table:
CREATE FUNCTION del_rows(p_table_name VARCHAR2)
RETURN NUMBER IS
BEGIN
  EXECUTE IMMEDIATE 'DELETE FROM '|| p_table_name;
  RETURN SQL%ROWCOUNT;
END;
/
BEGIN DBMS_OUTPUT.PUT_LINE(
  del_rows('EMPLOYEE_NAMES')|| ' rows deleted.');
END;
/
```

Beispiel Methode 2: NDS mit DML-Anweisung

```
-- Insert a row into a table with two columns:
CREATE PROCEDURE add_row(p_table_name VARCHAR2,
   p_id NUMBER, p_name VARCHAR2) IS
BEGIN
  EXECUTE IMMEDIATE 'INSERT INTO '|| p_table_name ||
        ' VALUES (:1, :2)' USING p_id, p_name;
END;
```

- Verwendung der USING-Klausel für die Parameter
- Die Zuordnung der BIND_Variablen erfolgt anhand der Position

Beispiel Methode 3: NDS mit Single-Row Query:

```
CREATE FUNCTION get_emp(p_emp_id NUMBER)
RETURN employees%ROWTYPE IS
  v_stmt VARCHAR2(200);
  v_emprec employees%ROWTYPE;
BEGIN
  v_stmt := 'SELECT * FROM employees ' ||
            'WHERE employee_id = :p_emp_id';
  EXECUTE IMMEDIATE v_stmt INTO v_emprec USING p_emp_id;
  RETURN v_emprec;
END;
/
DECLARE
  v_emprec employees%ROWTYPE := get_emp(100);
BEGIN
  DBMS_OUTPUT.PUT_LINE('Emp: '|| v_emprec.last_name);
END;
/
```

Beispiel Methode 3: NDS mit Multi-Row Query:

```
-- Use OPEN-FOR, FETCH, and CLOSE processing:

CREATE PROCEDURE list_employees( p_deptid NUMBER default null )
    IS
  TYPE emp_refcsr_type IS REF CURSOR;
  cur_emp  emp_refcsr_type;
  rec_emp  employees%ROWTYPE;
  v_stmt varchar2(200) := 'SELECT * FROM employees';
BEGIN
  IF p_deptid IS NULL THEN OPEN cur_emp FOR v_stmt;
  ELSE
    v_stmt := v_stmt || ' WHERE department_id = :id';
    OPEN cur_emp FOR v_stmt USING p_deptid;
  END IF;
  LOOP
    FETCH cur_emp INTO rec_emp;
    EXIT WHEN cur_emp%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(rec_emp.department_id||
                         ' ' || rec_emp.last_name);
  END LOOP;
  CLOSE cur_emp;
END;
```

Deklaration von Cursor-Variablen

```
-- Declare two cursor types as REF CURSORs:

CREATE OR REPLACE FUNCTION process_data
RETURN employees%ROWTYPE IS
  TYPE cur_ref_type     IS REF CURSOR; -- weak ref. cursor
  TYPE cur_ref_emp_type IS REF CURSOR; -- weak ref. cursor
 . . .

-- Declare a cursor variable using the cursor type:
  v_dept_csr cur_ref_type;
  v_emp_csr  cur_ref_emp_type;

BEGIN
  OPEN v_dept_csr FOR SELECT * FROM departments;
      OPEN v_emp_csr FOR SELECT  * FROM employees;

-- Then use as normal cursors
. . .

END;
```

Beispiel: NDS mit Ausführung eines anonymen PL/SQL-Blocks

```
CREATE FUNCTION annual_sal( p_emp_id NUMBER)
RETURN NUMBER IS
  v_plsql varchar2(200) :=
    'DECLARE '||
    ' rec_emp employees%ROWTYPE; '||
    'BEGIN '||
    ' rec_emp := get_emp(:empid); ' ||
    ' :res := rec_emp.salary  * 12; ' ||
    'END;';
  v_result NUMBER;
BEGIN
 EXECUTE IMMEDIATE v_plsql
         USING IN p_emp_id, OUT v_result;
  RETURN v_result;
END;
/
EXECUTE DBMS_OUTPUT.PUT_LINE(annual_sal(100))
```

Nutzung des Pakets `DBMS_SQL`

- `DBMS_SQL` muss verwendet werden, falls die Anzahl der Eingabe- oder Ausgabe-Variablen der SQL-Anweisung vorab nicht bekannt ist.
- Ansonsten gilt: Meist ist NDS einfacher zu nutzen und hat auch die bessere Performanz.
- Beispiele für die Nutzung von `DBMS_SQL` :
  - Unbekannte `SELECT` -Liste zur Übersetzungszeit
  - Unbekannte Anzahl an Spalten einer `SELECT` -Anweisung oder wir kennen deren Typen nicht.

Bestandteile von DBMS_SQL

Enthält Funktionen und Prozeduren für:

- OPEN_CURSOR
- PARSE
- BIND_VARIABLE
- EXECUTE
- FETCH_ROWS
- CLOSE_CURSOR

DBMS_SQL mit einer DML-Anweisung:

```
CREATE OR REPLACE FUNCTION delete_all_rows
   (p_table_name  VARCHAR2) RETURN NUMBER IS
    v_cur_id      INTEGER;
    v_rows_del    NUMBER;
BEGIN
   v_cur_id := DBMS_SQL.OPEN_CURSOR;
   DBMS_SQL.PARSE(v_cur_id,
     'DELETE FROM '|| p_table_name, DBMS_SQL.NATIVE);
   v_rows_del := DBMS_SQL.EXECUTE (v_cur_id);
   DBMS_SQL.CLOSE_CURSOR(v_cur_id);
   RETURN v_rows_del;
END;
/
```

```
CREATE TABLE temp_emp AS SELECT * FROM employees;
BEGIN
 DBMS_OUTPUT.PUT_LINE('Rows Deleted: ' ||
delete_all_rows('temp_emp'));
END;/
```

DBMS_SQL mit einer parametrisierten DML-Anweisung

```
CREATE PROCEDURE insert_row (p_table_name VARCHAR2,
 p_id VARCHAR2, p_name VARCHAR2, p_region NUMBER) IS
  v_cur_id       INTEGER;
  v_stmt         VARCHAR2(200);
  v_rows_added   NUMBER;
BEGIN
  v_stmt := 'INSERT INTO '|| p_table_name ||
            ' VALUES (:cid, :cname, :rid)';
  v_cur_id := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQL.PARSE(v_cur_id, v_stmt, DBMS_SQL.NATIVE);
  DBMS_SQL.BIND_VARIABLE(v_cur_id, ':cid', p_id);
  DBMS_SQL.BIND_VARIABLE(v_cur_id, ':cname', p_name);
  DBMS_SQL.BIND_VARIABLE(v_cur_id, ':rid', p_region);
  rows_added := DBMS_SQL.EXECUTE(v_cur_id);
  DBMS_SQL.CLOSE_CURSOR(v_cur_id);
  DBMS_OUTPUT.PUT_LINE(v_rows_added||' row added');
END;/
```

Zusammenspiel mit NDS

Interoperabilität zwischen NDS und `DBMS_SQL` wird in Oracle 11g unterstützt

- SQL-Anweisungen länger als 32 KB sind erlaubt in NDS
- `DBMS_SQL.PARSE()` ist überladen für `CLOB`s.
- `REF CURSOR` kann zu `DBMS_SQL` Cursor und umgekehrt konvertiert werden.
- `DBMS_SQL` unterstützt alle Datentypen einschließlich Collections und Objekttypen.