Objects used in Examples on Oracle User Defined SQL Sources

Example1 (user defined function as a table collection expression):

In this example, user-defined SQL function **return_dept_data()** returns records from EMP table where employee DEPTNO matches with the given P_DEPTNO.

In Oracle, the result set returned by this type of function is treated as a table.

```
DROP FUNCTION return_dept_data;
DROP TYPE emp_tab_typ;
DROP TYPE emp_typ;

CREATE OR REPLACE TYPE emp_typ as object (
  empno NUMBER(4),
  job VARCHAR2(9),
  sal NUMBER(7,2)
);
/

CREATE OR REPLACE TYPE emp_tab_typ AS TABLE OF emp_typ;
/

CREATE OR REPLACE FUNCTION return_dept_data( p_deptno IN NUMBER )
RETURN emp_tab_typ AS
  v_ret   emp_tab_typ;
BEGIN
  SELECT
  CAST(MULTISET(SELECT empno,job,sal FROM emp WHERE deptno = p_deptno)
  AS emp_tab_typ)
  INTO v_ret
  FROM dual;
  RETURN v_ret;
END return_dept_data;
/
```

Following SQL SELECT statement can be used in DMExpress as a table source:

```
SELECT * FROM TABLE(return_dept_data(10))
```

Example2 (nested table column as a table collection expression):

In this example, CUSTOMER table has ADDRESS column which is a nested table column. A nested table column in Oracle can be as simple as a composite column with two or more fields (or) a table by itself.

In Oracle, a nested table column must have a user-defined type explicitly created for the layout of the fields. In this example ADDRESS_TYP is created first, and then it is used in the definition of the CUSTOMER table to define the composite column ADDRESS.

```
DROP TABLE customer;
DROP TYPE address_tab_typ;
DROP TYPE address_typ;

CREATE TYPE  address_typ
AS OBJECT (
   street1  VARCHAR2(30),
   street2  VARCHAR2(30),
   city     VARCHAR2(30),
   state    VARCHAR2(30),
   zip      VARCHAR2(5),
   zip_ext  VARCHAR2(4));
/

CREATE TYPE address_tab_typ AS TABLE OF address_typ;
/


CREATE TABLE customer (
   customer_id    NUMBER(10),
   lastname       VARCHAR2(30),
   firstname      VARCHAR2(30),
   address        address_tab_typ)
   NESTED TABLE   address STORE AS address_stor_tab;
```

Using the INSERT statements given below, first we insert the data into regular table columns and then insert the data into nested table columns.

We are first inserting the data into regular columns of CUSTOMER table.

```
INSERT INTO customer VALUES (1001, 'Barak', 'Obama', address_tab_typ());
INSERT INTO customer VALUES (1002, 'Clooney', 'George', address_tab_typ());
```

We are now inserting the data into nested table column ADDRESS for customers with CUSTOMER_ID 1001 and 1002.

```
INSERT INTO TABLE(SELECT t1.address FROM customer t1 WHERE t1.customer_id =
1001) VALUES ('1600 Pennsylvania Avenue NW', NULL,
'Washington','D.C','20500',NULL);

INSERT INTO TABLE(SELECT t1.address FROM customer t1 WHERE t1.customer_id =
1002) VALUES ('123 Main St', 'APT 123', 'Hollywood','CA','12345',NULL);

COMMIT;
```

Now the following SQL SELECT statement can be used in DMExpress as a table source. It returns CUSTOMER_ID, LASTNAME, FIRSTNAME, and associated ADDRESS of the customer, by joining CUSTOMER table with nested ADDRESS table:

```
SELECT t1.customer_id, t1.lastname, t1.firstname, t2.*
FROM customer t1, TABLE(t1.address) t2
```

Example3 (query that calls a user-defined function):

In this example, user-defined SQL function get_bonus() returns the COMM from the BONUS table for the employee with the given employee name.

```
CREATE OR REPLACE FUNCTION get_bonus(v_ename in varchar2)
RETURN NUMBER AS
v_bonus NUMBER;
BEGIN
 SELECT comm INTO v_bonus FROM bonus WHERE ename = v_ename;
 RETURN v_bonus;
END get_bonus;
/
```

The following SQL SELECT statement can be used in DMExpress as a table source:

```
SELECT empno, ename, get_bonus(ename) AS emp_bonus, sal FROM emp
```