How to Pass an Array of Object Type To Static Function of That Type

Solution shorter than the title

by Przemysław Kruglej
(11-2013)

przemyslawkruglej.com
przemyslaw.kruglej@gmail.com

Table of Contents

1 Array of an Object Type Argument in a Function of That Object Type

Interesting question regarding *Object Types* was asked not a long ago on StackOverflow:

http://stackoverflow.com/questions/19271570/how-to-type-a-static-function-paramater-as-a-table-of-objects

What author was trying to accomplish was to declare a *static function* in an object type which would take as a parameter an *array of objects* of the *object type in which it was declared*.

Is that even possible? *(dramatic pause)*

1.1 First Try

Let's check firstly if we can define a non-array parameter of the same object type:

```
CREATE OR REPLACE
TYPE my_object AS OBJECT (
  some_number NUMBER,
  STATIC FUNCTION static_test_function(
                                    p_my_object IN my_object) RETURN NUMBER
);
/

> TYPE MY_OBJECT compiled
```

Alright, no errors – now let's create a nested table type of `my_object` type:

```
CREATE TYPE my_objects_tab_t IS TABLE OF my_object;
/

TYPE MY_OBJECTS_TAB_T compiled
```

So far, so good. Next, let's alter declaration of `my_object`'s static function to take `my_objects_tab_t` as an argument:

```
CREATE OR REPLACE
TYPE my_object AS OBJECT (
  some_number NUMBER,
  STATIC FUNCTION static_test_function(
                        p_my_objects_tab IN my_objects_tab_t) RETURN NUMBER
);
/

Error report:
02303. 00000 -  "cannot drop or replace a type with type or table dependents"
```

Yeah, this is not going to work – we are not allowed to change the definition of our object type because there is a dependent nested table type. So, is this it? Fear not, we have another option.

1.2 Data of ANYDATA Datatype as Argument's Type. Ouch.

Oracle offers a type called `ANYDATA`, which, is a self-describing type – it stores both an instance of some other type and its description. It can even be persisted into a table.

So how can this undecided fellow help us?

We can define the argument of the static function as `ANYDATA`, and then, pass as an actual argument an instance of `ANYDATA` containing an array of objects of our type! Then, inside our static function,

we will use a member function of `ANYDATA` type called `GetCollection`, which will return the objects it holds – in this case, an array of objects of our type.

Below is a simple presentation of this solution.

1.2.1 Step 1: Declare our object type again

```
CREATE OR REPLACE
TYPE my_object AS OBJECT (
  some_number NUMBER,
  STATIC FUNCTION static_test_function(
                              p_my_objects_tab IN ANYDATA) RETURN NUMBER
);
/

> TYPE MY_OBJECT compiled
```

Notice that now the type of the argument is `ANYDATA`.

1.2.2 Step 2: Create nested table type

```
CREATE TYPE my_objects_tab_t AS TABLE OF my_object;
/

> TYPE MY_OBJECTS_TAB_T compiled
```

1.2.3 Step 3: Extract my_objects_tab_t object from ANYDATA

```
CREATE OR REPLACE
TYPE BODY my_object AS

  STATIC FUNCTION static_test_function(
                                  p_my_objects_tab IN ANYDATA) RETURN NUMBER
  AS
    v_my_objects_tab my_objects_tab_t;
    v_dummy NUMBER;
    v_result NUMBER := 0;
  BEGIN
    -- get the collection from ANYDATA object
    --   v_dummy holds status (succes or not)
    v_dummy := p_my_objects_tab.GetCollection(v_my_objects_tab);

    -- as an example, loop through the list and sum up the some_number field
    FOR v_i IN v_my_objects_tab.FIRST..v_my_objects_tab.LAST
    LOOP
      v_result := v_result + v_my_objects_tab(v_i).some_number;
    END LOOP;

    RETURN v_result;
  END;
END;
/

> TYPE BODY MY_OBJECT compiled
```

This is where all the magic happens. In line `#13`, we use the `GetCollection` member function of `ANYDATA` type, which takes as an argument a collection which should be filled with elements held in `ANYDATA` object. Then, as a test, we loop through elements of our just retrieved nested table and sum

up `some_number` field of each element.

1.2.4 Step 4: Test it!

```
DECLARE
  v_my_objects_tab my_objects_tab_t;
  v_anydata_with_my_objects_tab ANYDATA;

  v_result NUMBER;
BEGIN
  -- create the collection and populate it with some sample data
  v_my_objects_tab := my_objects_tab_t(my_object(5), my_object(10));

  -- create ANYDATA object from the collection
  v_anydata_with_my_objects_tab :=
                              ANYDATA.ConvertCollection(v_my_objects_tab);

  -- call the static function expecting ANYDATA parameter
  v_result := my_object.static_test_function(v_anydata_with_my_objects_tab);

  dbms_output.put_line(v_result);
END;
/

> 15
```

Number "15" appears in the standard output – seems like it works! In the third line, we define an `ANYDATA` variable which will contain nested table object. In lines `#11` and `#12`, we utilize the `ConvertCollection` function of `ANYDATA` type, which returns an instance of `ANYDATA` type containing the collection passed as the argument.

1.3 Am I an apple? Cause I smell like an orange

You may wonder: what's going to happen if we pass `ANYDATA` object holding something that we do not expect?

```
CREATE TYPE my_str_tab_t IS TABLE OF VARCHAR2(100);
/

DECLARE
  v_my_str_tab my_str_tab_t;
  v_fake ANYDATA;

  v_result NUMBER;
BEGIN
  v_my_str_tab := my_str_tab_t('A', 'B');

  v_fake := ANYDATA.ConvertCollection(v_my_str_tab);

  v_result := my_object.static_test_function(v_fake);

  dbms_output.put_line(v_result);
END;
/

22626. 00000 -  "Type Mismatch while constructing or accessing OCIAnyData"
*Cause:    Type supplied is not matching the type of the AnyData.
           If piece wise construction or access is being attempted, the
```

```
          type supplied is not matching the type of the current attribute.
```

As you might have expected, we got an error – we wanted nested table of our `my_object` type, and we got strings instead. Can we somehow prevent ourselves from such situations? Not only can we, but we should!

We can check the type of object held within `ANYDATA` using the `GetTypeName` function. Let's improve our static function:

```
CREATE OR REPLACE
TYPE BODY my_object AS
  STATIC FUNCTION static_test_function(p_my_objects_tab IN ANYDATA) RETURN
NUMBER
  AS
    v_my_objects_tab my_objects_tab_t;
    v_dummy NUMBER;
    v_result NUMBER := 0;
  BEGIN
    IF p_my_objects_tab.GetTypeName != 'PK.MY_OBJECTS_TAB_T' THEN
      dbms_output.put_line('Expected "PK.MY_OBJECTS_TAB_T" type, got: ' ||
        p_my_objects_tab.GetTypeName);
      RETURN NULL;
    END IF;

    -- get the collection from ANYDATA object
    --   v_dummy holds status (succes or not)
    v_dummy := p_my_objects_tab.GetCollection(v_my_objects_tab);

    -- as an example, loop through the list and sum up the some_number field
    FOR v_i IN v_my_objects_tab.FIRST..v_my_objects_tab.LAST
    LOOP
      v_result := v_result + v_my_objects_tab(v_i).some_number;
    END LOOP;

    RETURN v_result;
  END;
END;
/

> TYPE BODY MY_OBJECT compiled
```

In line `#10`, we use the `GetTypeName` function to check if the type of object inside `ANYDATA` argument is the one we expect. If not, we print the name of the type we got and return `NULL`. `GetTypeName` returns fully qualified name of the type, i. e. with the schema that owns it.

Let's see if this will prevent us from error this time:

```
DECLARE
  v_my_str_tab my_str_tab_t;
  v_fake ANYDATA;

  v_result NUMBER;
BEGIN
  v_my_str_tab := my_str_tab_t('A', 'B');

  v_fake := ANYDATA.ConvertCollection(v_my_str_tab);

  v_result := my_object.static_test_function(v_fake);
```

```
  dbms_output.put_line(v_result);
END;
/

Expected "PK.MY_OBJECTS_TAB_T" type, got: PK.MY_STR_TAB_T
```

And it did. We see the type we got – `PK.MY_STR_TAB_T`.

2 Further Reading & Useful Links

As it turned out, the trick was to use the `ANYDATA` type. If you would like to read more about it, you'll find a link below.

ANYDATA in Oracle Documentation:

- http://docs.oracle.com/cd/B28359_01/appdev.111/b28419/t_anydat.htm

---

I hope you enjoyed my article. If you have found any errors in it (even typos), you think that I haven't explained anything clearly enough or you have an idea how I could make the article better – please, do not hesitate to contact me, or leave a comment.