Basics of Calling PL/SQL Subroutines from PHP using OCI

Calling a PL/SQL stored procedure from PHP using OCI

To call a **PL/SQL stored procedure** using OCI:

- the string containing the procedure call must begin with **`BEGIN`** and end with **`END;`**
  - (yes, ending with a semicolon WITHIN the string, unlike for a SQL statement string!)
- and between those, you put your desired PL/SQL procedure call **also** followed by a semicolon,
- putting bind variables as desired for the procedure's arguments.

For example, if you have a PL/SQL stored procedure **`do_stuff`** that expects two arguments, you could write an OCI **`oci_parse`** call like this (assuming `$conn` contains a connection object from a successful call to `oci_connect`):

```
$proc_stmt = oci_parse($conn,
                       "BEGIN do_stuff(:bind_var1, :bind_var2); END;");
```

Then:

- Use **`oci_bind_by_name`** to give a value to each bind variable involved -- for example:

  ```
  oci_bind_by_name($proc_stmt, ":bind_var1", $desired_arg1);
  oci_bind_by_name($proc_tmt, ":bind_var2", $desired_arg2);
  ```

- Use **`oci_execute`** to execute the resulting statement object:

  ```
  oci_execute($proc_stmt, OCI_DEFAULT);
  ```

- Depending on the procedure, if the procedure changed the database, then when the logical transaction including this procedure call is deemed to be successfully completed, be sure to explicitly commit the resulting database state:

  ```
  oci_commit($conn);
  ```

Calling a PL/SQL stored function from PHP using OCI

The interesting wrinkle when calling a PL/SQL stored function from PHP using OCI is that, as when you call a PL/SQL stored function from SQL*Plus' `SQL>`, you are **REQUIRED** to provide a "place" to put

the value returned by the function.

So -- you MUST write the function call as a SQL*Plus assignment statement, including an *output* bind variable to hold the result.

That is, to call a **PL/SQL stored function** using OCI:

- the string containing the procedure call still must begin with **`BEGIN`** and end with **`END;`**
  - (yes, ending with a semicolon WITHIN the string, unlike for a SQL statement string!)
- and between those, you put a bind variable to hold the function's result

  then the assignment operator `:=`

  then the desired PL/SQL function call **also** followed by a semicolon,
- also putting bind variables as desired for the function's arguments.

For example, if you have a PL/SQL stored function **`get_stuff`** that expects two arguments, you could write an OCI **`oci_parse`** call like this (assuming `$conn` contains a connection object from a successful call to `oci_connect`):

```
$funct_stmt = oci_parse($conn,
    "BEGIN :result_var := get_stuff(:bind_var1, :bind_var2); END;");
```

Then:

- Use **`oci_bind_by_name`** to give a value to each (input) bind variable involved -- for example:

  ```
  oci_bind_by_name($funct_stmt, ":bind_var1", $desired_arg1);
  oci_bind_by_name($funct_stmt, ":bind_var2", $desired_arg2);
  ```

  - But -- what about that bind variable `:result_var`?

    You use **`oci_bind_by_name`** here, also, but for an "output" bind variable the third argument here specifies what PHP *variable* is to be **set** to what the function call **returns**!

    - That is, instead of giving a third argument giving the value to put ***into*** the bind variable,

      here you give a third argument that is a variable to be **set** to what the output bind variable's value **becomes**...!

    And, you need a ***fourth*** argument, giving the maximum length of the value that might be put into that PHP variable.

    According to Example #10 at https://www.php.net/manual/en/function.oci-bind-by-name.php, "The default type will be a string type ..." so using a maximum length of, say, 10 would mean at most 10

characters or at most 10 digits would be returned.

- That is, based on what you expect the function to return, it should be reasonable to base this maximum length on the maximum number of characters or digits you expect in the value returned by the function.

So -- say that `get_stuff` returns an integer guaranteed to be less than 10,000 (less than or equal to 9999). Then this would work:

```
oci_bind_by_name($funct_stmt, ":result_var", $stuff_val, 4);
```

- Use **`oci_execute`** to execute the resulting statement object:

```
oci_execute($funct_stmt, OCI_DEFAULT);
```

  - And because of the **`oci_bind_by_name`** statement above for **`:result_var`**, at *this* point the PHP variable **`$stuff_val`** will be set to whatever this function call now returns.

    (And as long as this value is 9999 or less, based on the maximum length specified of 4, all will be well!)

- (A typical function would not change the database state -- but if you ever have an unusual one that does, then when the logical transaction including this function call is deemed to be successfully completed, be sure to explicitly commit the resulting database state with **`oci_commit`**.)