Using Oracle Database 10 *g* PL/SQL New Features

Purpose

This module describes the new Oracle Database 10g features introduced in PL/SQL.

Topics

This module will discuss the following:

Viewing Screenshots

Overview

Back to List of Topics

In Oracle Database 10 *g* , several enhancements are available in the PL/SQL language. This lesson covers:

1. Bulk Binding Enhancements
2. Debugging PL/SQL Using JDeveloper
3. PL/SQL Compiler Enhancements

Bulk Binding Enhancements

Back to List

The Oracle Database 10 *g* extends upon the enhancements introduced in Oracle9 *i* , in the area of Bulk Binding. The `SAVE EXCEPTIONS` syntax was introduced in Oracle9 *i* to capture exceptions while bulk inserting (deleting or updating) rows. While this feature allows the DML to continue, saving any exceptions in the `SQL%BULK_EXCEPTIONS` collection, the performance of the operation is greatly affected. In a certain scenario where the collection being processed is sparse due to the application logic involved, this would be an unnecessary overhead. In Oracle Database 10 *g* , you can overcome these issues with the new `INDICES OF` or `VALUES OF` features.

Debugging PL/SQL Using JDeveloper

Back to List

Starting version 9.0.3, JDeveloper supports PL/SQL debugging. This is achieved with the Java Debugging Wire Protocol (JDWP). With Oracle9 *i* Release 2 and higher, the debugger uses the industry standard JPDA (Java Platform Debugger Architecture) whereas for prior releases (Oracle8 *i* or Oracle9 *i* Release 1) the `DBMS_DEBUG` package is used. The debugging UI gives you the ability to single step, step into, and step over PL/SQL objects as well as the ability to set PL/SQL expressions in the Watch and Inspector window, set conditional breakpoints, and view collection data.

PL/SQL Compiler Enhancements

Back to List

The PL/SQL compiler is substantially improved in the Oracle Database 10 *g* . The back end code generator is re-engineered. Also, a new parameter, called `PLSQL_OPTIMIZE_LEVEL` , has been introduced to speed up execution of PL/SQL code. This is in addition to Native compilation (which is available since Oracle9 *i*). Currently the `PLSQL_OPTIMIZE_LEVEL` parameter can be set to " `1` " or " `2` ". `1` is the default, `2` increases the performance.

Prerequisites

Back to List

Before starting this module, you should have performed the following:

- ☐ **1.** Completed the Configuring Linux for the Installation of Oracle Database 10g lesson
- ☐ **2.** Completed the Installing the Oracle Database 10g on Linux lesson
- ☐ **3.** Completed the Installing Oracle9i JDeveloper on Linux lesson.
- ☐ **4.** Download and unzip the plsql.zip into your working directory.

Examining Bulk Binding Using `INDICES OF` and `VALUES OF` Keywords

Back to List

The following example illustrates the use of `INDICES OF` and `VALUES OF` keywords when using Bulk Binding in PL/SQL.

The `INDICES OF` keyword can be used in a scenario where a collection of records (which is dense) is validated programmatically and invalid records (those not meeting the specified criterion) are removed from the collection, resulting in a sparse collection of valid elements which must then be bulk inserted into a table. By using the `INDICES OF` the exceptions for the missing records are not generated.

The `VALUES OF` keyword can be used in a scenario where a collection of records (sparse or dense) must be copied to one or more collection variables, based upon some condition whereby certain records may or may not be copied, and then inserted into a table. This can be efficiently done with the use of the " `VALUES OF` " syntax and using a pointer array

whose elements are pointers to the selected records within the original collection. This reduces the need to create multiple copies of data. Exception handling is done per pointer record, which means that if two or more pointer records are pointing to the same "original" data, any exceptions reported will indicate the iteration number (pointer element number).

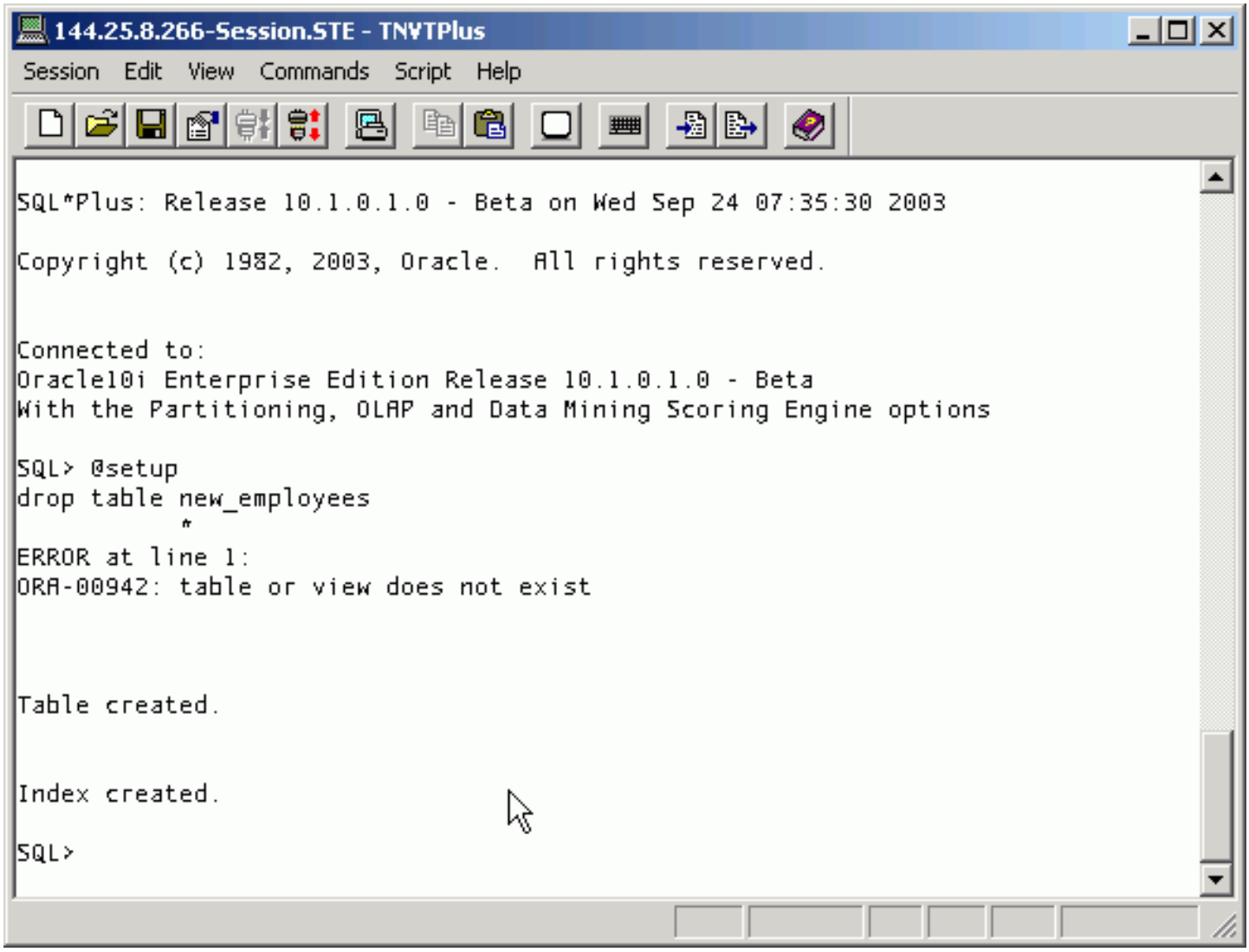
This example uses the EMPLOYEES table to build a collection of employees that is deliberately sparse.

☐ **1.** You will first need to create a NEW_EMPLOYEES table from the existing EMPLOYEES table (in the hr schema) and create a unique index on EMPLOYEE_ID column. This will generate an exception when inserting a duplicate record. Execute the following commands from a terminal window:

```
cd wkdir
sqlplus hr/hr
@setup.sql
```

The **setup.sql** script contains the following:

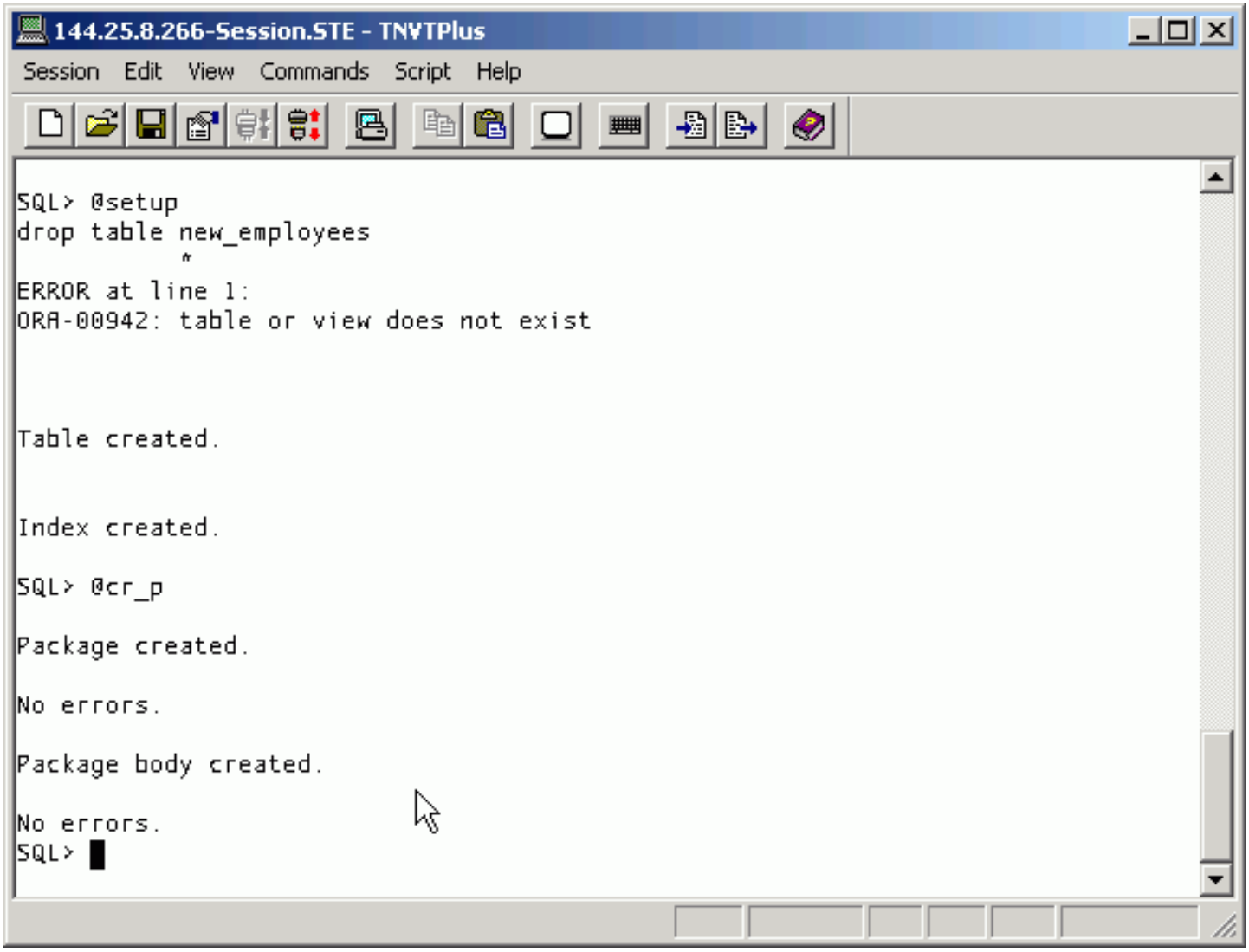
```
drop table new_employees;
create table new_employees as select * from employees where 1=2;
create unique index new_employees_employee_id
  on new_employees (employee_id);
```



☐ **2.** Next, you need to create the package `P_BULK_BIND` . This package contains various procedures which create the sparse collection and bulk insert the records into the `NEW_EMPLOYEES` table using three different approaches.

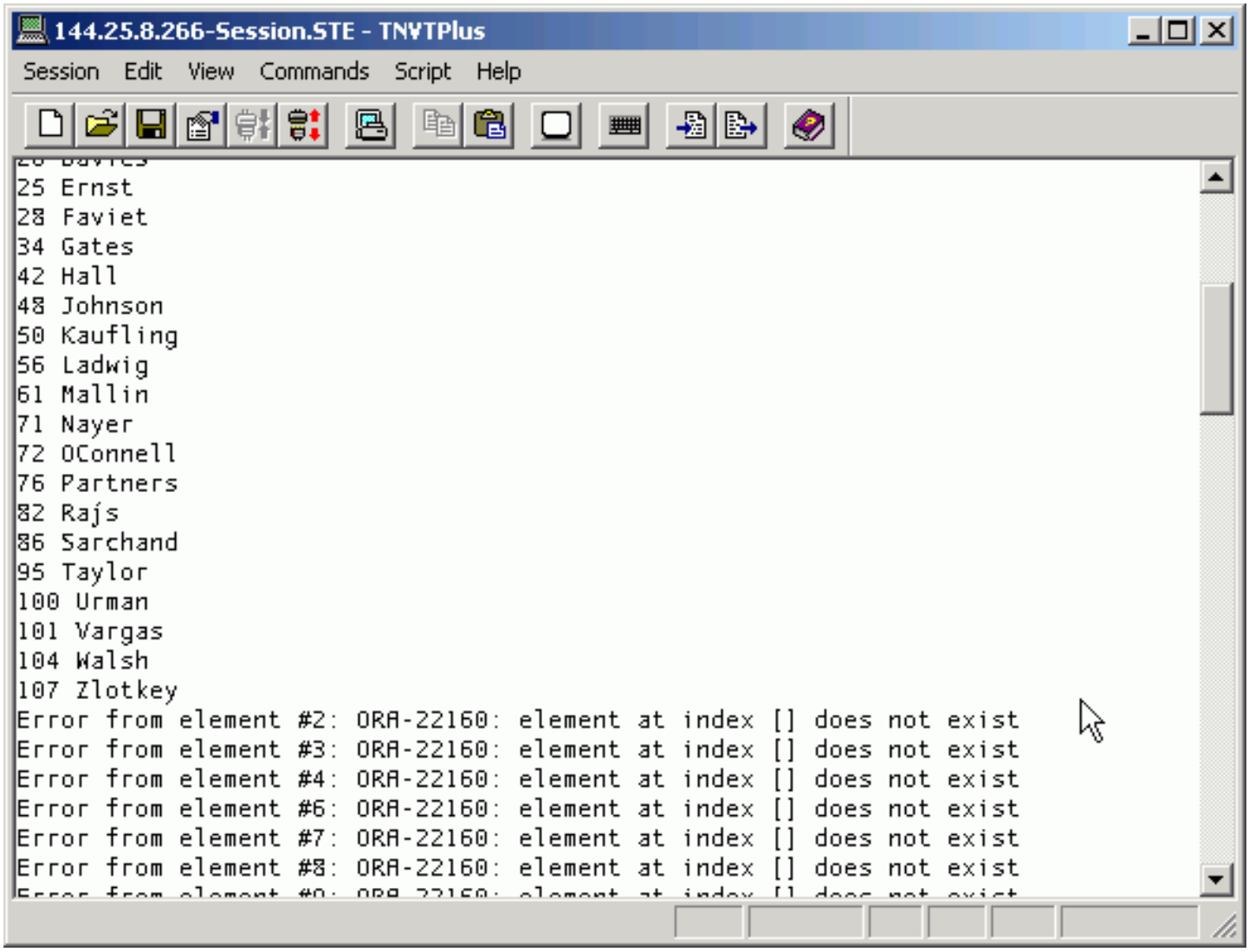
`@cr_p.sql`

Click here to see what is contained in the **`cr_p.sql`** script.



☐ **3.** To see how the Bulk Binding works in Oracle Database 10g, you will first see how it was done in Oracle9i. The `FORALL` statement uses the `SAVE EXCEPTIONS` syntax to suppress exceptions raised due to missing records, which are later printed in the `EXCEPTION` block. Execute the following commands:

```
set serverout on size 100000
exec p_bulk_bind.Bulk_Insert_Pre_10g (i_make_sparse => true)
```



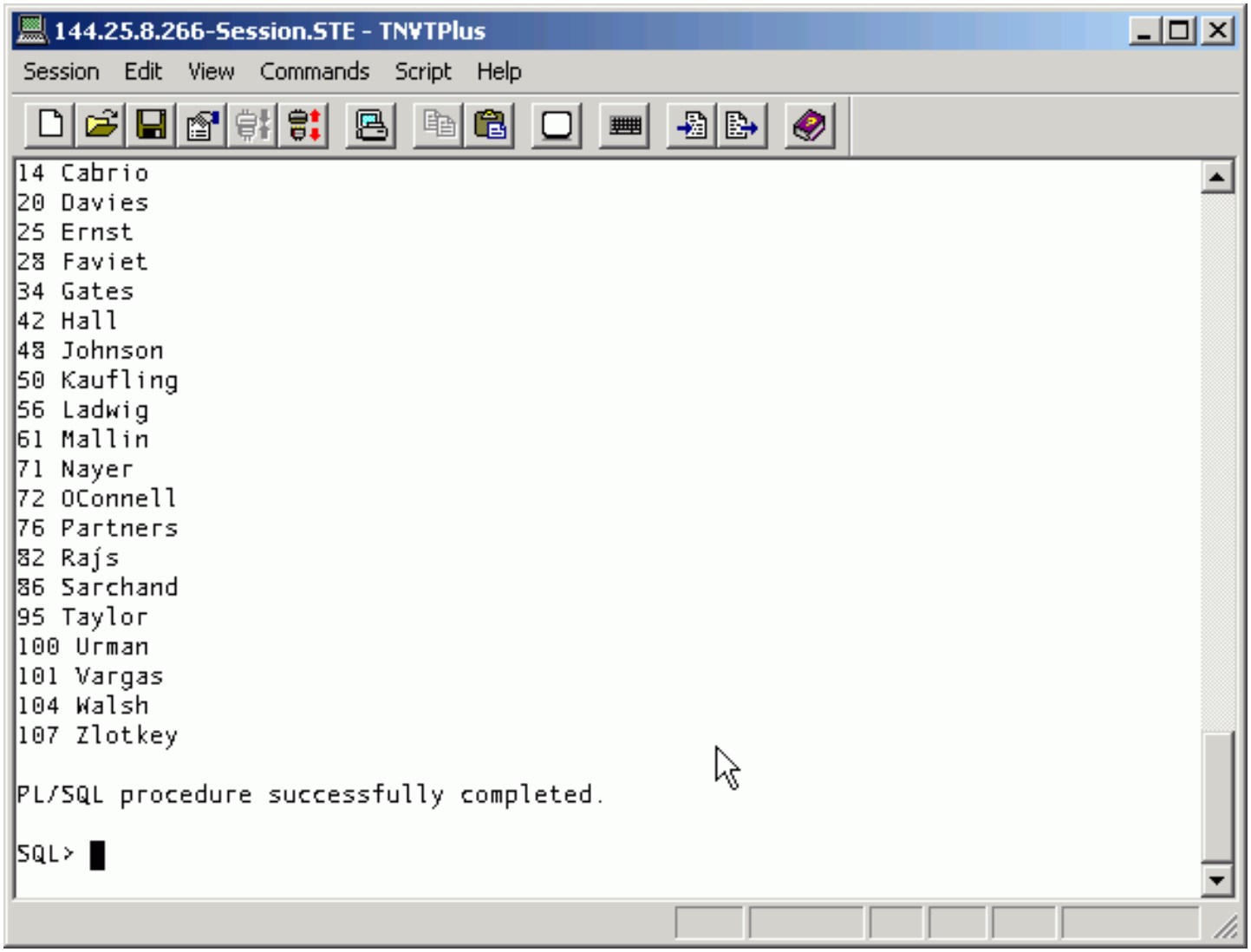

```
-- this code is located in the Bulk_Insert_Pre_10g procedure
...


forall j in g_emp_recs.First()..g_emp_recs.Last()

save exceptions
  insert into new_employees values g_emp_recs(j);


exception when bulk_errors then
  for j in 1..sql%bulk_exceptions.Count()
  loop
    Dbms_Output.Put_Line ( 'Error from element #' ||
    To_Char(sql%bulk_exceptions(j).error_index) || ': ' ||
    Sqlerrm(SQL%bulk_exceptions(j).error_code) );
  end loop;

...
```

From the output, you see that each missing record has generated an exception (`ORA-22160`). As mentioned earlier, even though the remaining records are processed via `SAVE EXCEPTIONS` syntax, this results in a performance degradation if the number of deleted records is fairly high.

☐ **4.** The next step is to use the Oracle Database 10 *g* `INDICES OF` syntax to ignore the missing records and thereby obtain much better performance. Execute the following commands:

```
exec p_bulk_bind.Bulk_Insert_With_Indices_Of -
   (i_cause_exception => false)
```

In this case the Boolean input value is used to ignore a block of code responsible for creating an exception condition as you will see in the next step.

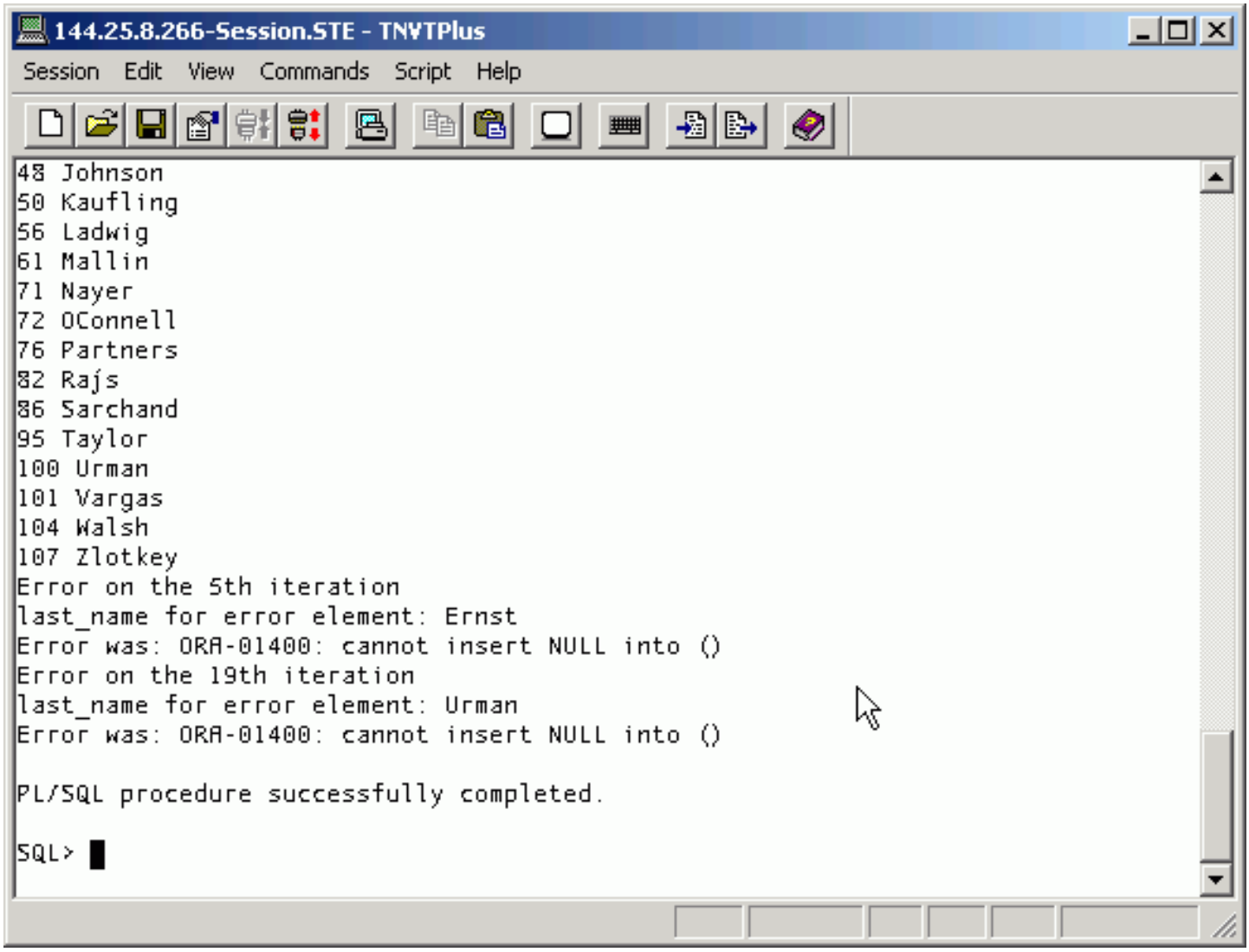


```
-- this code is located in the
Bulk_Insert_With_Indices_Of

-- procedure


...

forall j in
indices of
 g_emp_recs
save exceptions
  insert into new_employees values g_emp_recs(j);


...
```

This time there are no exceptions raised due to the missing records.

☐ **5.** The above procedure can also be run with the input parameter set to true. This would cause an exception while inserting which must be trapped (by `SAVE EXCEPTIONS`) and later processed. The condition causing the exception is simulated via a procedure call, which nulls the 'email' columns for the employees with the last name of 'Ernst' and 'Urman'. Since the `EMAIL` column has a `NOT NULL` restriction, this causes an exception when the two corresponding records are inserted. The exceptions are later processed from the `SQL%BULK_EXCEPTIONS` collection. Execute the following commands:

```
exec p_bulk_bind.Bulk_Insert_With_Indices_Of -
   (i_cause_exception => true)
```



-- this code is located in the exception area
-- in the
`Bulk_Insert_With_Indices_Of`
procedure

```
...

forall j in
indices of
 g_emp_recs
-- between g_emp_recs.First() and g_emp_recs.Last()
-- optional

save exceptions
  insert into new_employees values g_emp_recs(j);



exception when bulk_errors then
  declare
    v_iteration pls_integer;
```

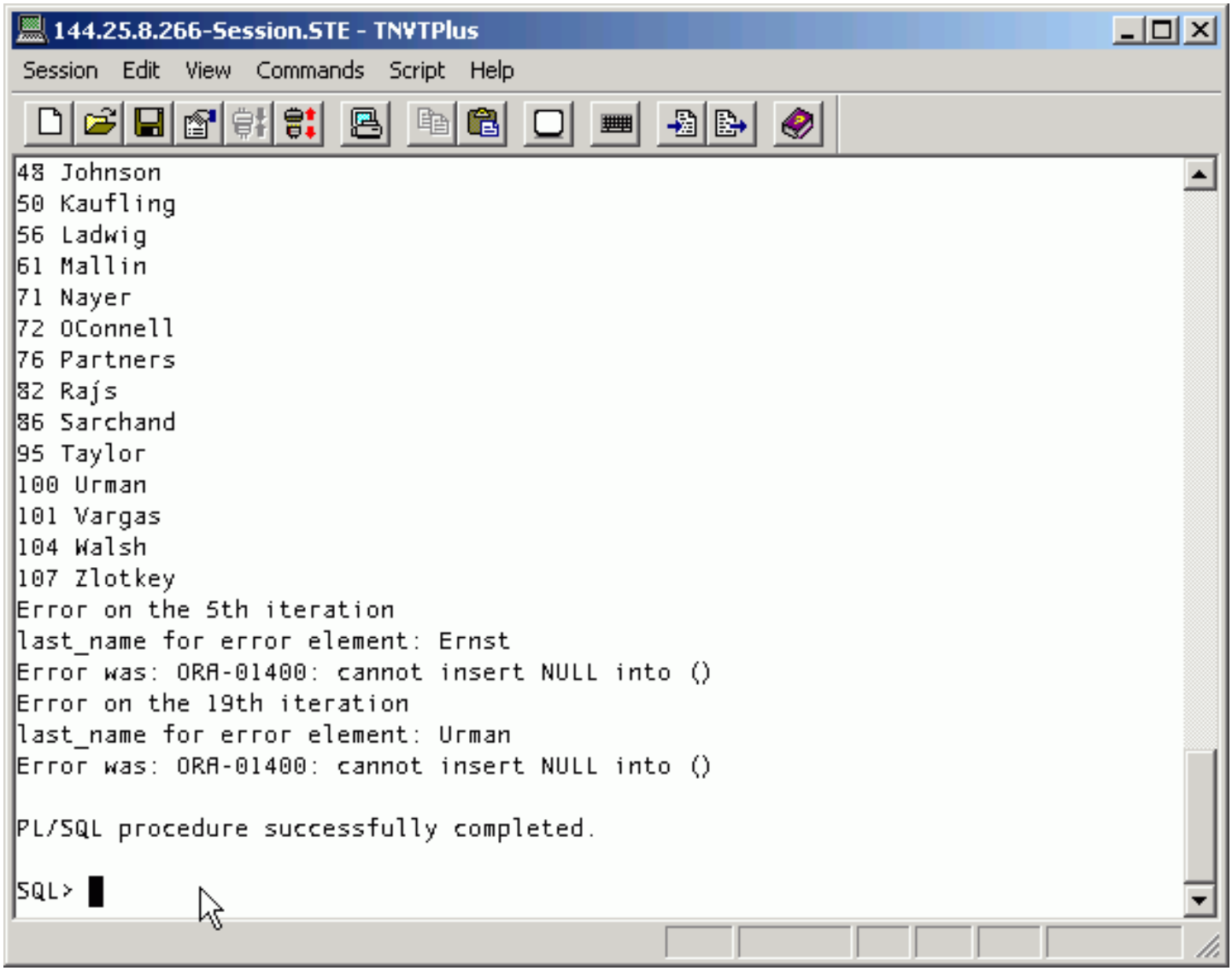
```
    n pls_integer;
    k pls_integer;
  begin
    for j in 1..
sql%bulk_exceptions
.Count()
    loop
      v_iteration := SQL%bulk_exceptions(j).error_index;
      Dbms_Output.Put_Line (
        'Error on the ' || To_Char(v_iteration)
         || 'th iteration' );

 ...
```

Note: The error is reported for the iteration number, this is new to the exception handling in Oracle Database 10 *g* .

☐ **6.** Another approach is to use `VALUES OF` clause to create a second collection of pointers to the elements of the original array. Execute the following commands:

```
exec p_bulk_bind.Bulk_Insert_With_Values_Of -
   (i_null_email => true, i_violate_pk => false)
```

The `Point_To_Sparse` procedure is used to create a collection of pointers based on the same logic as before, that is, it points to the elements with the first occurrence of each letter in the last name It is called from the procedure `Bulk_Insert_With_Values_Of` .



-- this code is located in the

`Bulk_Insert_With_Values_Of`

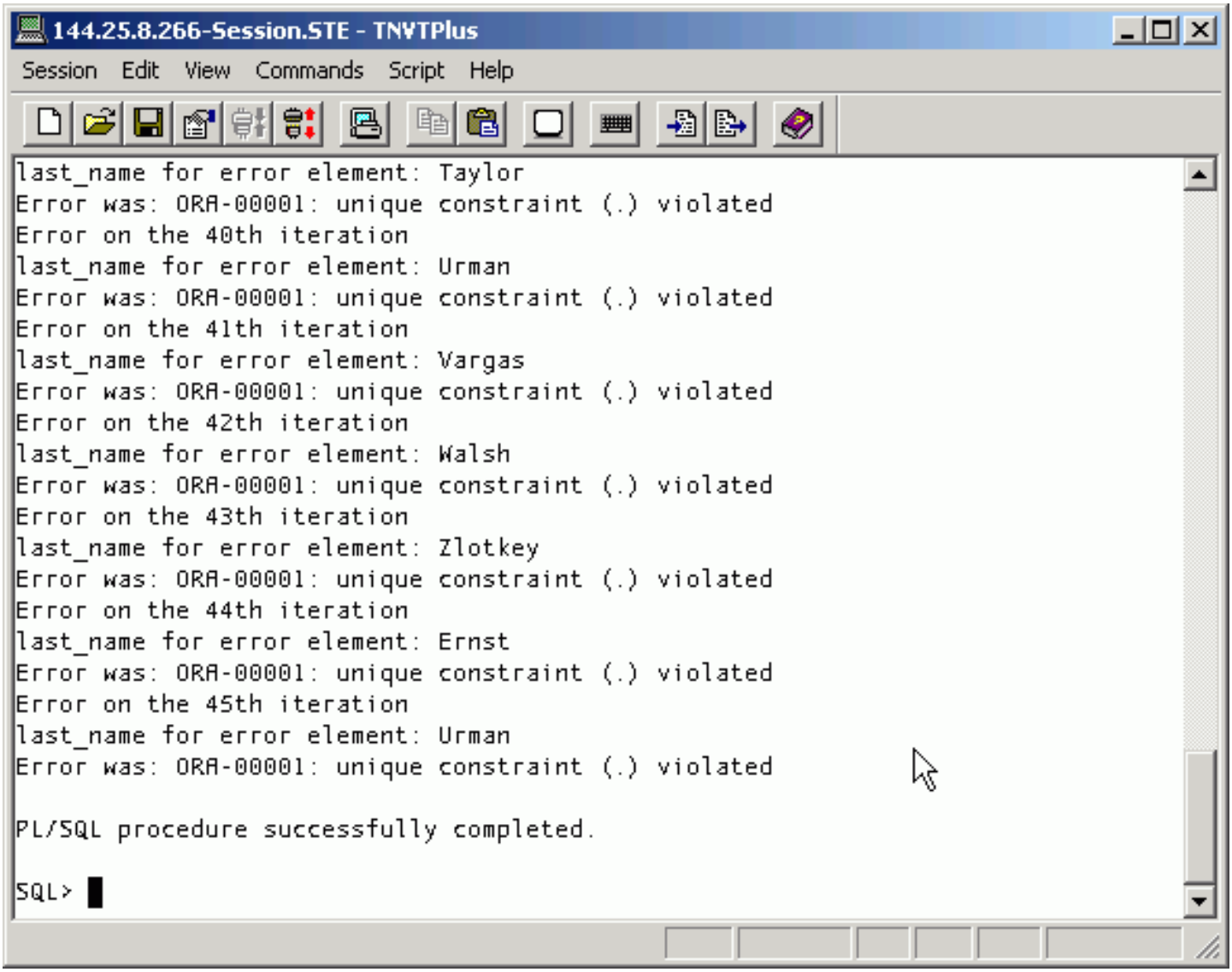
-- procedure

```
...

forall j in
values of
 g_values_of_tab
save exceptions
  insert into new_employees values g_emp_recs(j);
...
```

The result is the same as step 5, an exception is raised for the 5th and the 19th iteration.

☐ **7.** To see the effect of reporting the iteration number (or pointer element number) as opposed to the index of the erroneous record, you can run the above procedure with `i_violate_pk => true` . To simulate the error condition, the procedure `Cause_Exception_For_Values_Of` is used to insert a duplicate index entry from the bound collection (`G_EMP_RECS`) into the pointer collection (`G_VALUES_OF_TAB`). Due to this, the unique key constraint (created in the first step) is violated and the exception is raised upon bulk insert into the `NEW_EMPLOYEES` table. Execute the following commands:

```
exec p_bulk_bind.Bulk_Insert_With_Values_Of -
   (i_null_email => false, i_violate_pk => true)
```



-- this code is located in the

`Bulk_Insert_With_Values_Of`

```
-- procedure

...

exception when bulk_errors then
  declare
     v_iteration pls_integer;
   begin
     for j in 1..sql%bulk_exceptions.Count()
     loop
       v_iteration := SQL%bulk_exceptions(j).error_index;
       Dbms_Output.Put_Line (
         'Error on the ' || To_Char(v_iteration) ||
         'th iteration' );



-- Find the index of the offending element
-- from the iteration number
       Dbms_Output.Put_Line (
         'last_name for error element: ' ||
          g_emp_recs(g_values_of_tab(v_iteration)).last_name );


       Dbms_Output.Put_Line (
         'Error was: ' ||
          Sqlerrm(SQL%bulk_exceptions(j).error_code) );
     end loop;
   end;

...
```

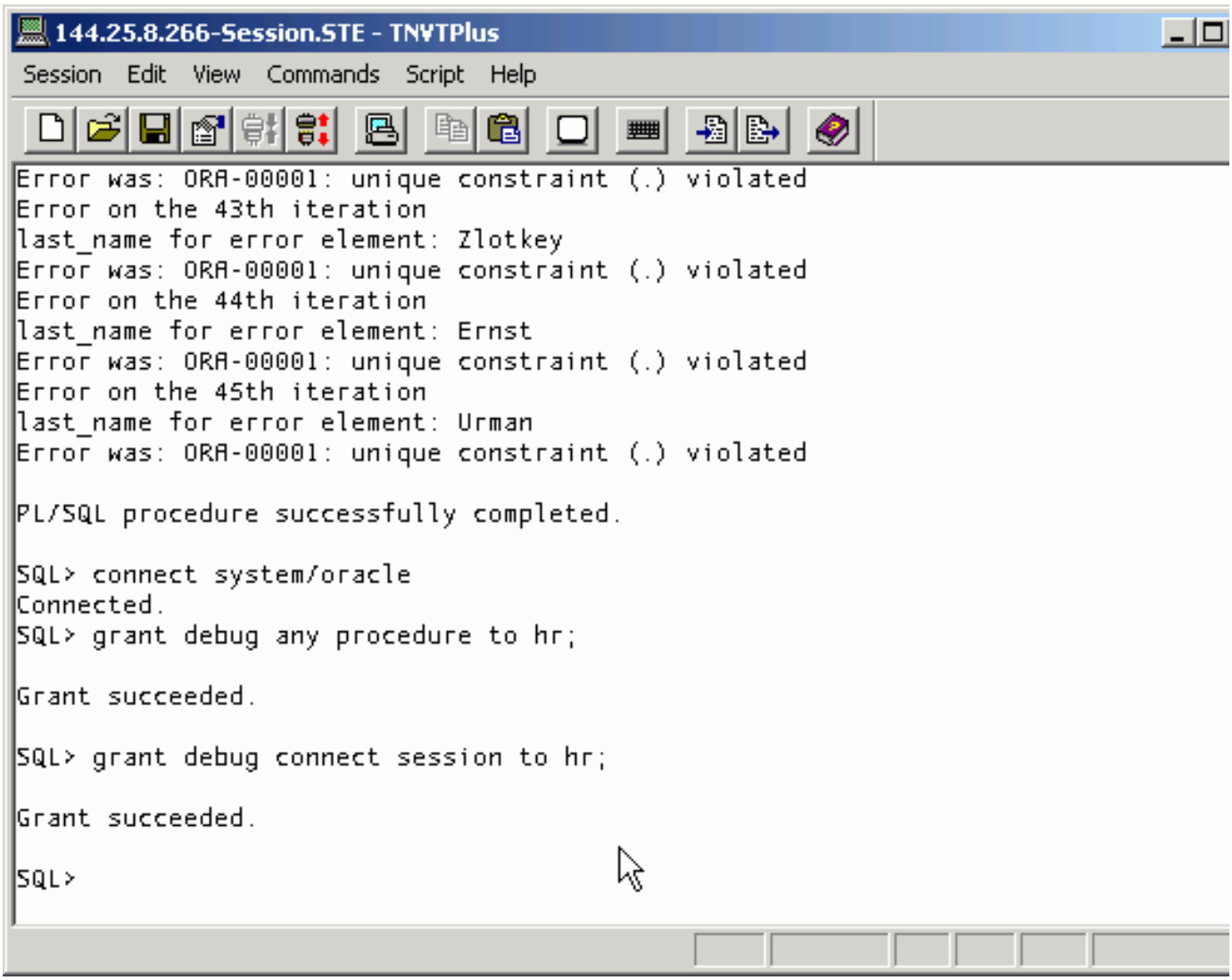
Setting Privileges and Parameters for Debugging

Back to List

Before you can use the PL/SQL debug capability within JDeveloper you need make sure the HR user has the appropriate privileges. Perform the following steps:

☐ **1.** Open a terminal window and execute the following:

```
cd wkdir
sqlplus system/<password>
grant debug any procedure to hr;
grant debug connect session to hr;
```



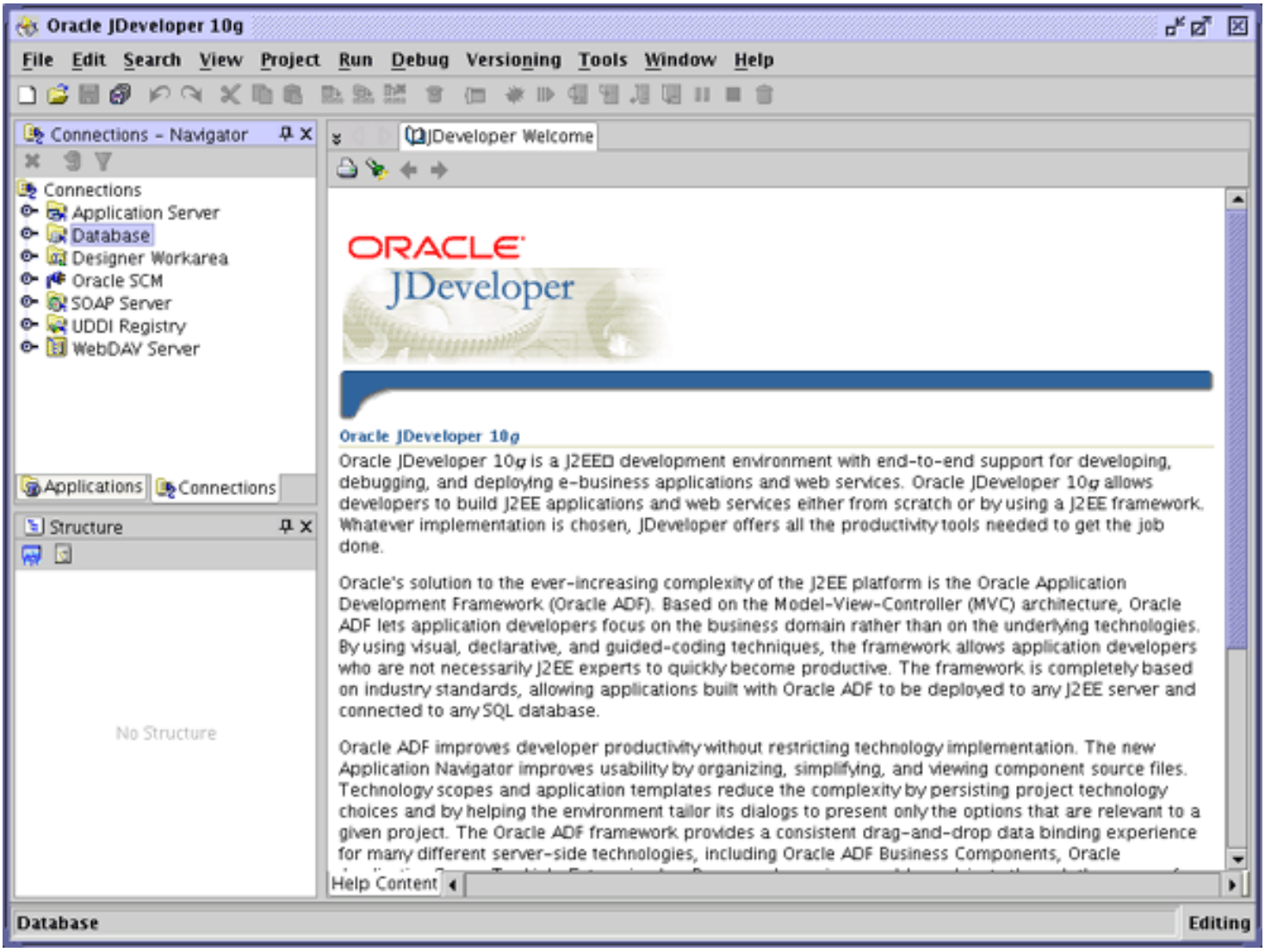
Creating a Database Connection

Back to List

In order to access the PL/SQL, you first need to create a Database Connection. Perform the following steps:

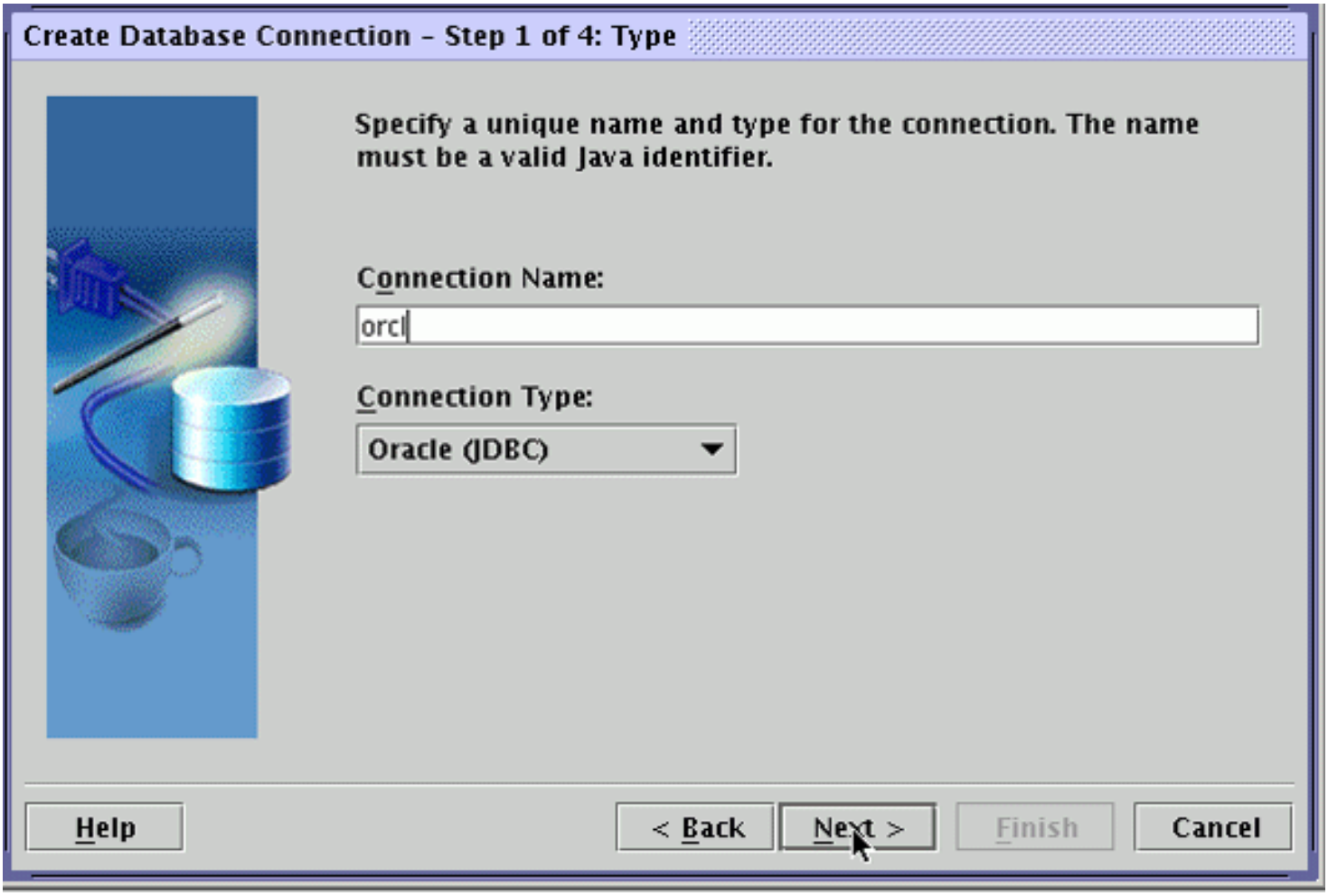
☐ **1.** Click on the JDeveloper icon you created in your panel.

☐ **2.** Select the **Connections** tab in the Applications - Navigation window. Then right click on **Database** and select **New Database Connection** .

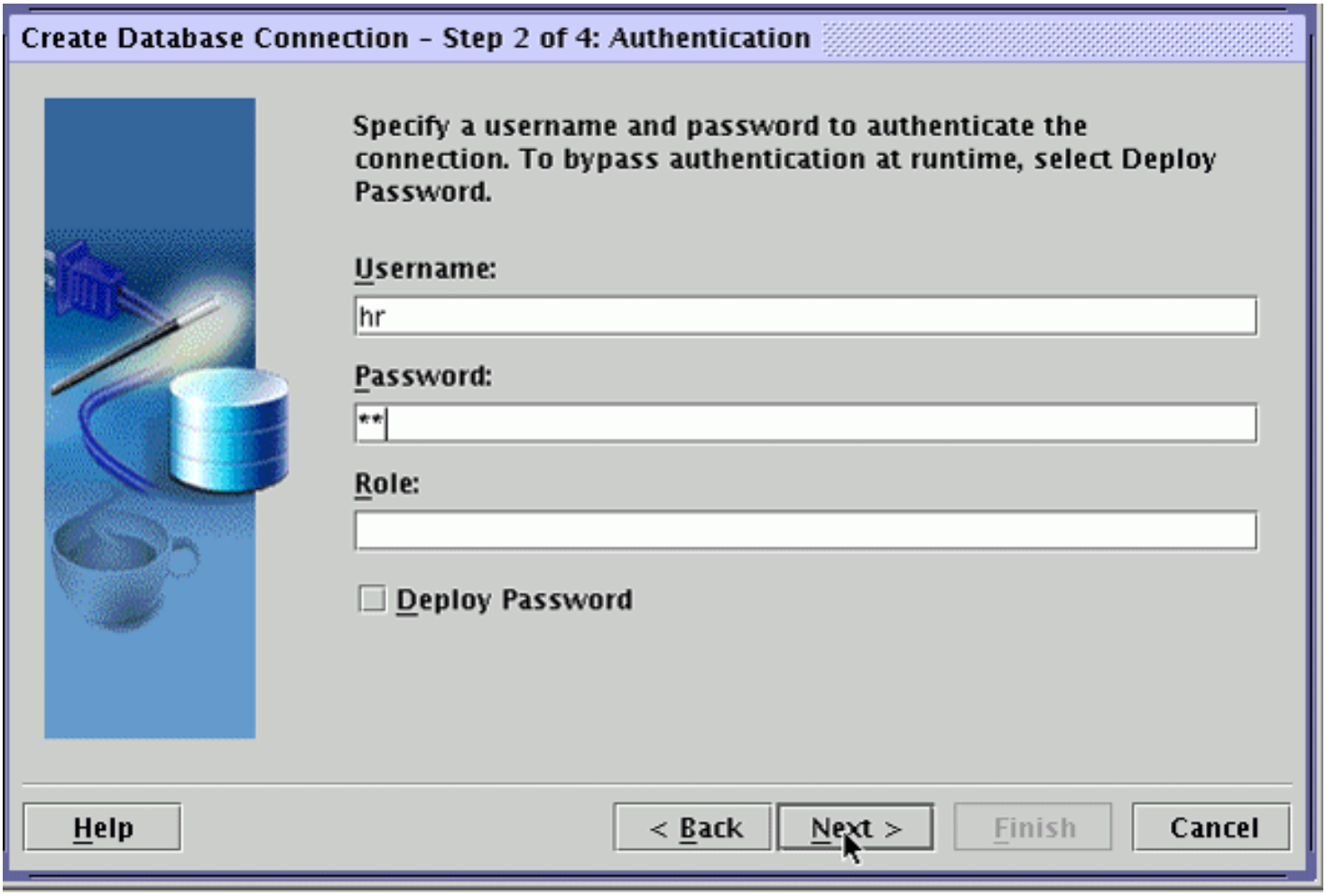


☐ **3.** At the Welcome window, click **Next** .

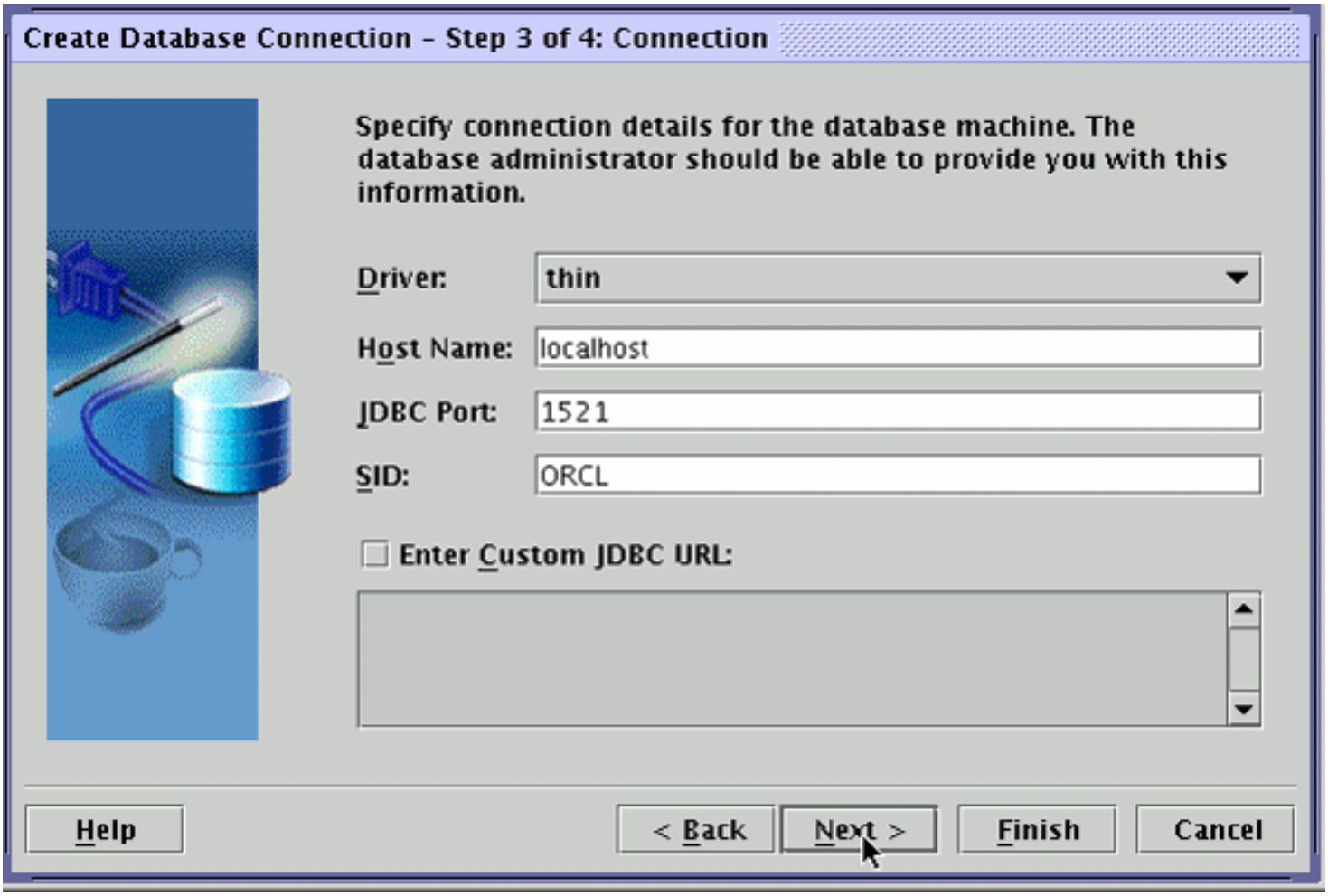
☐ **4.** Enter **orcl** for the Connection name and click **Next** .



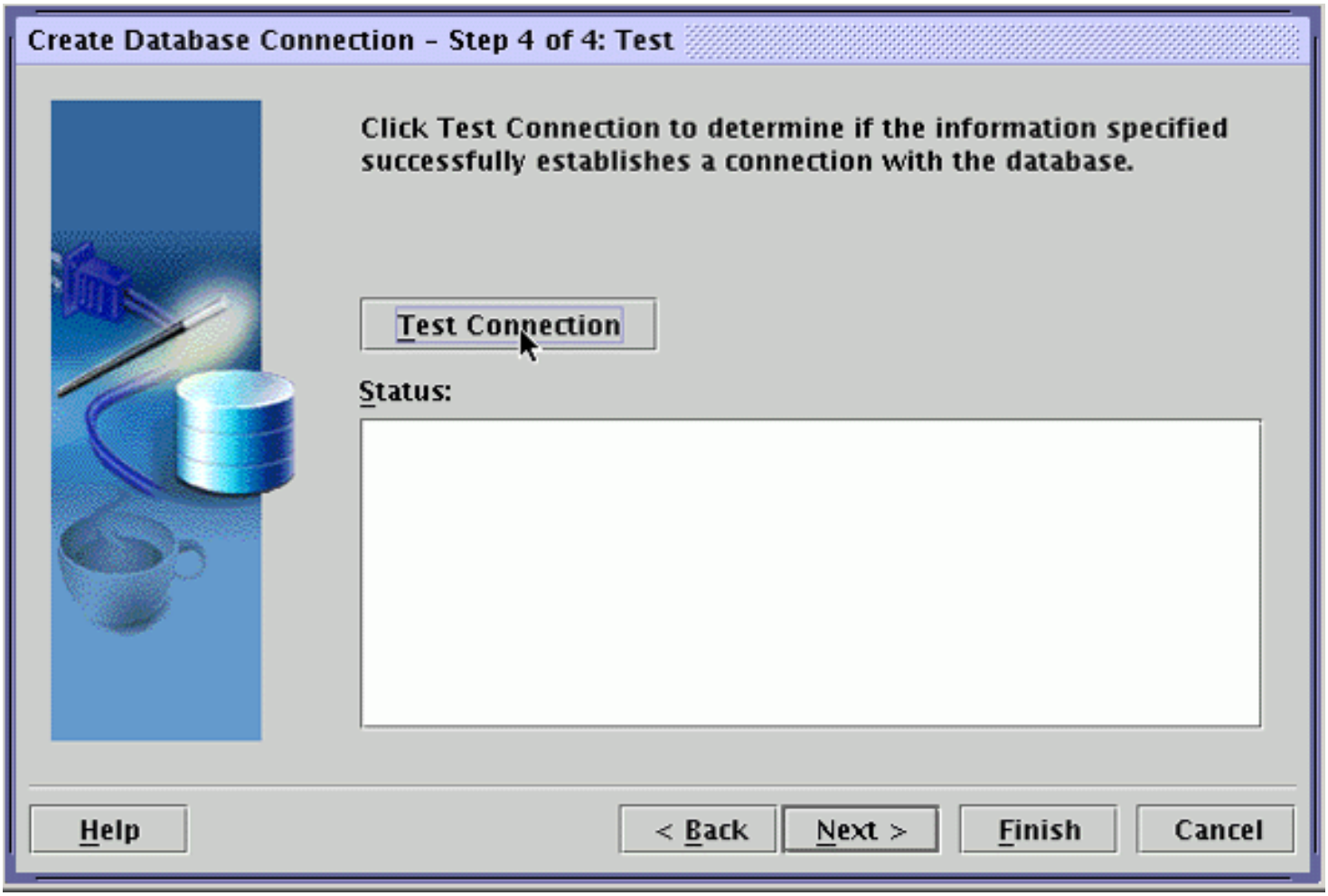
☐ **5.** Enter **hr** for the username and password and click **Next** .



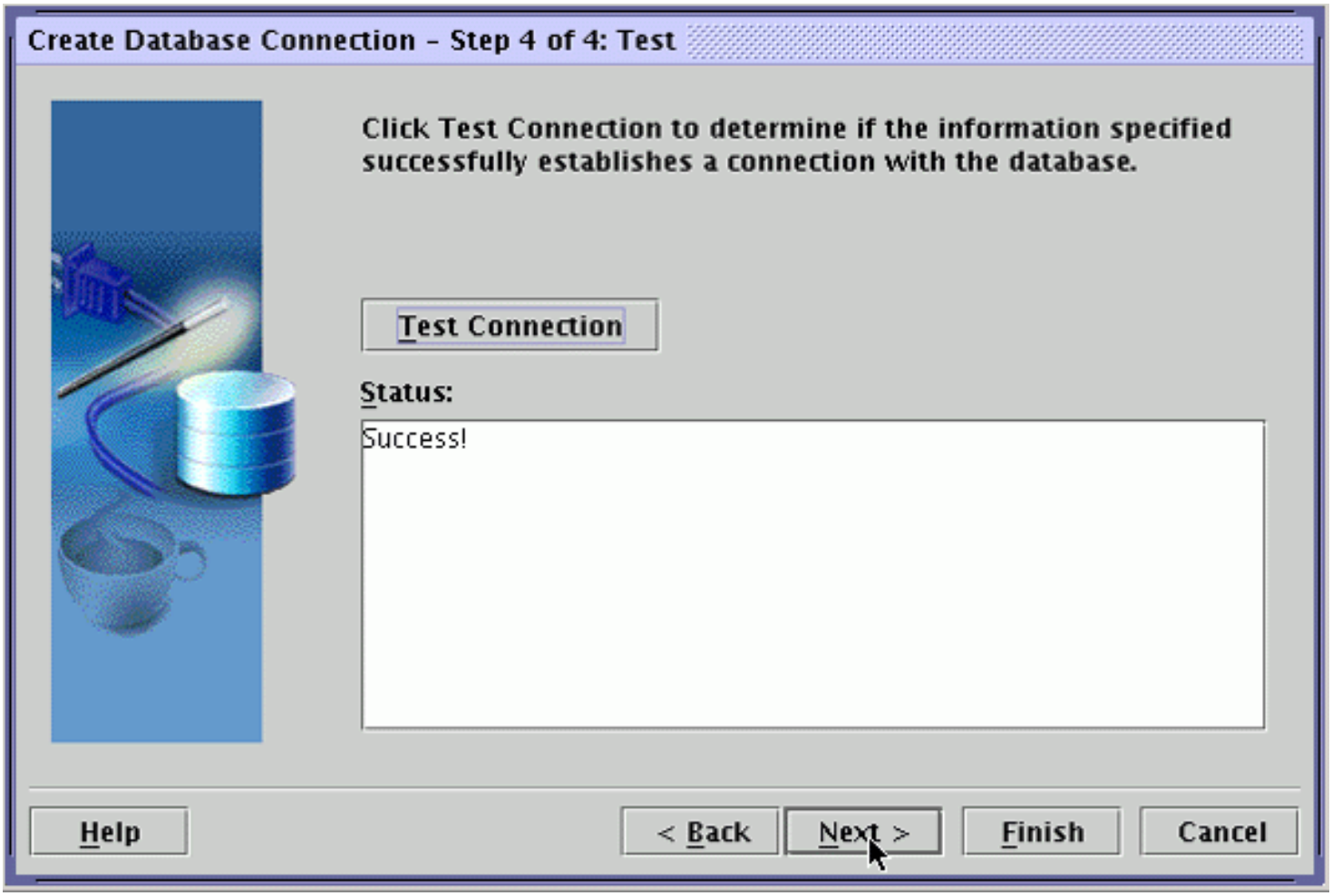
☐ **6.** The connection details for your database are shown. Accept the defaults and click **Next** .



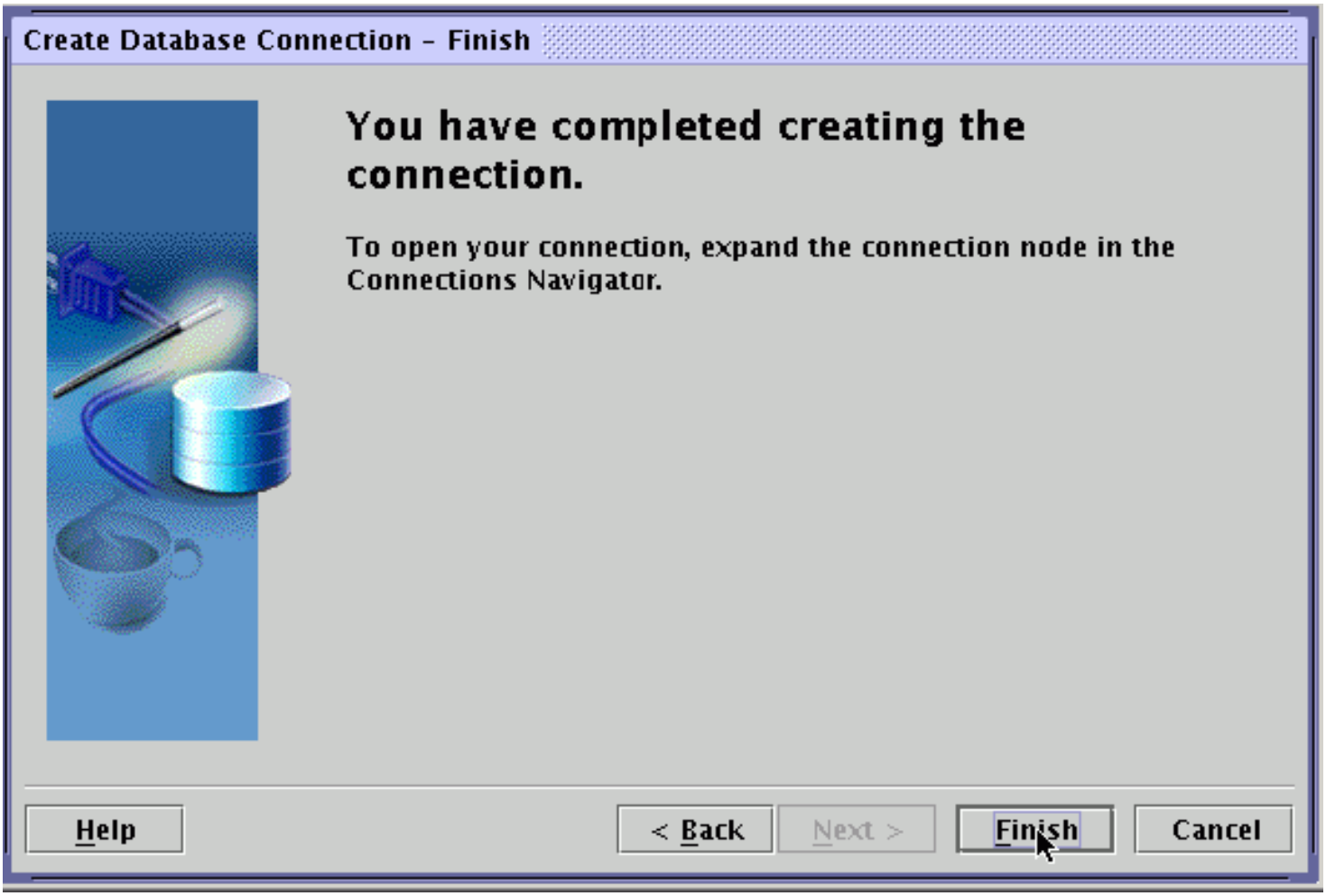
☐ **7.** Test your connection. Click **Test Connection** .



☐ **8.** When you receive the Success! message, click **Next** .



☐ **9.** Click **Finish** to create the connection.

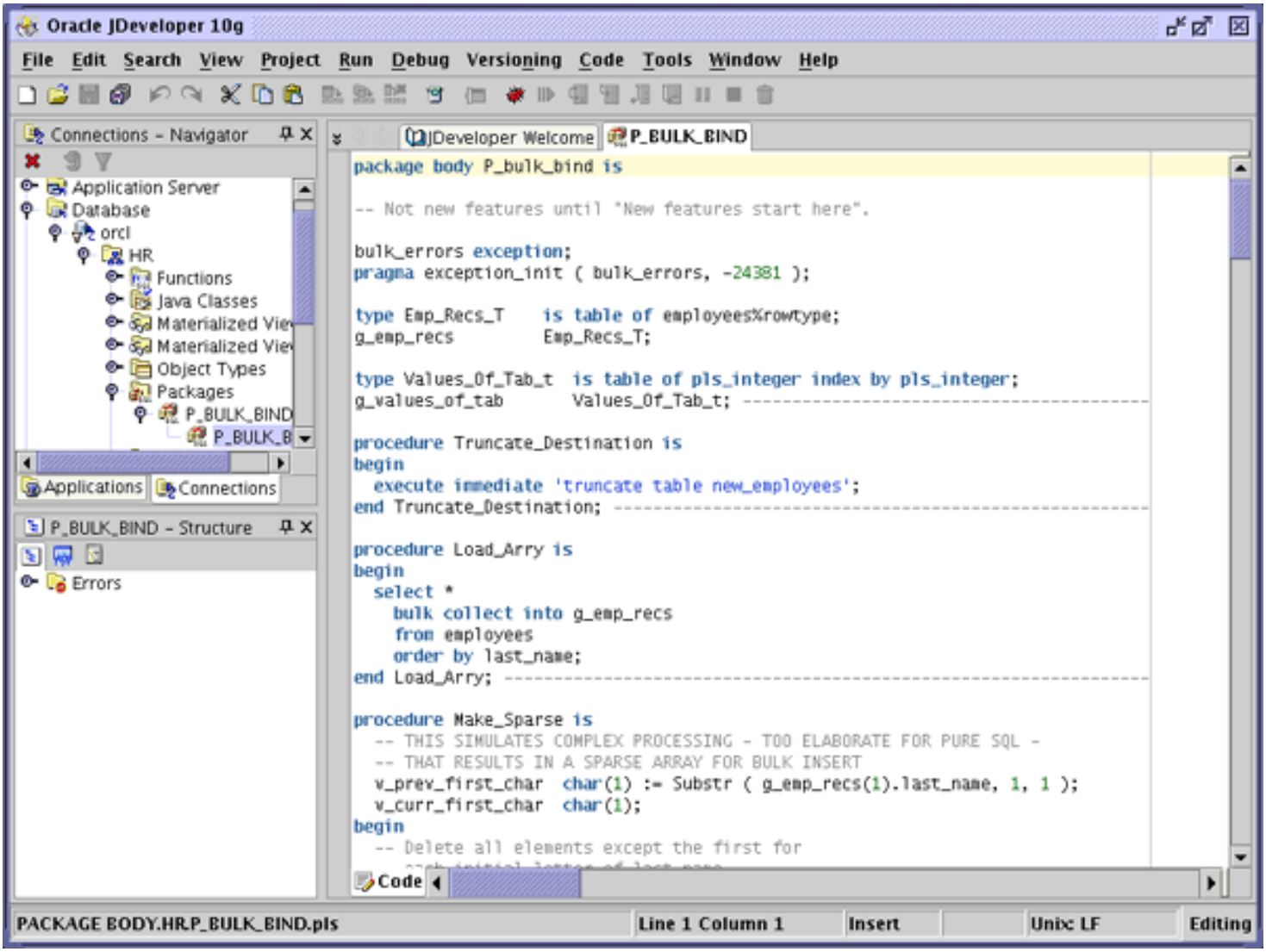


Debugging a PL/SQL Package Body Using JDeveloper

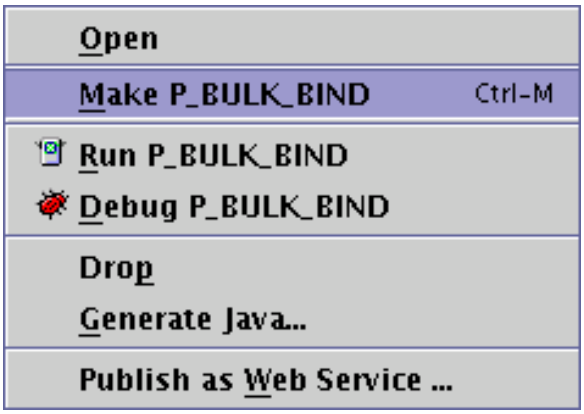
Back to List

You will run the PL/SQL package `P_BULK_BIND` you used previously using the JDeveloper PL/SQL Debugger. Perform the following:

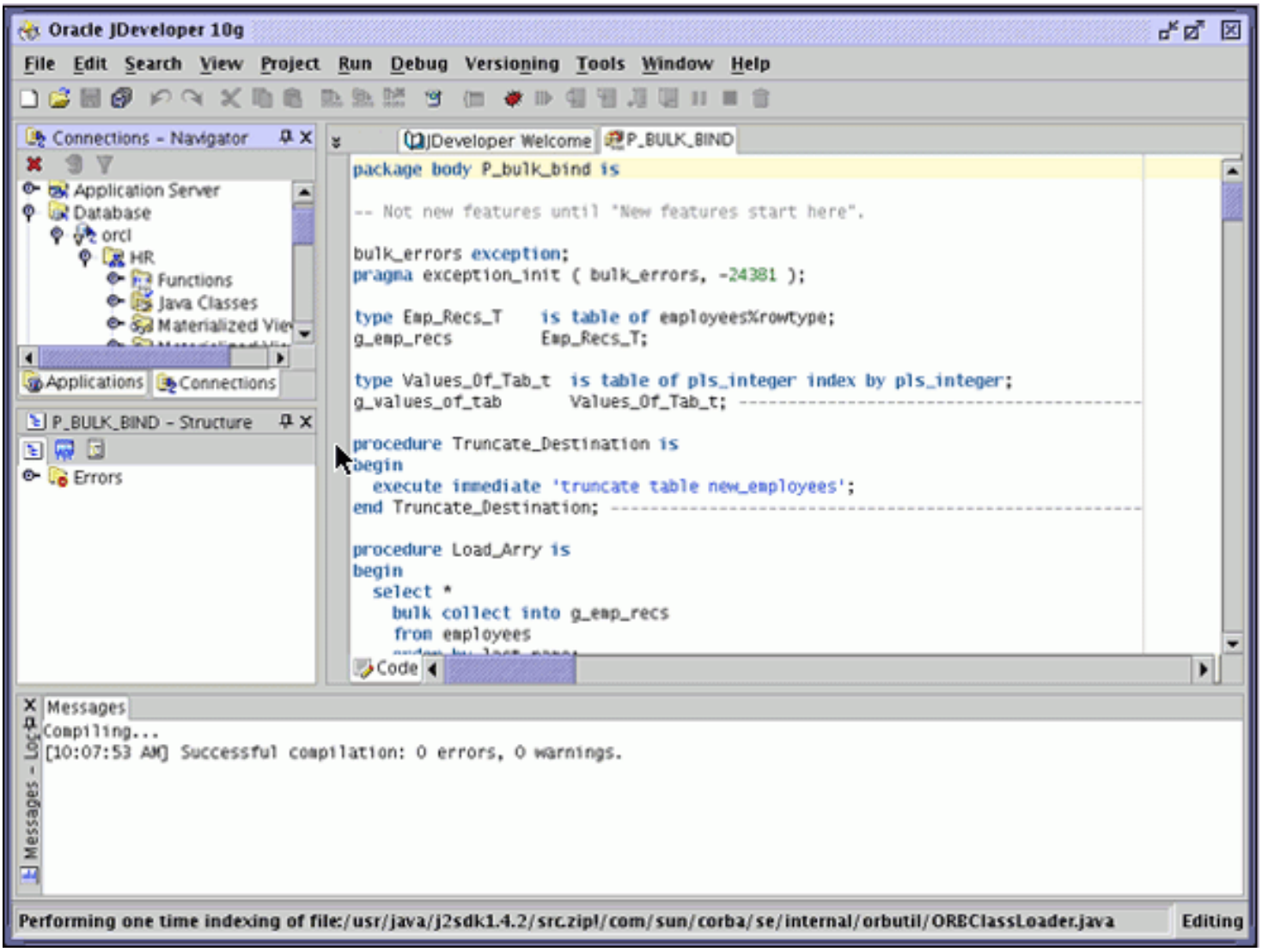
☐ **1.** From the Connections - Navigator window, expand **Connections -> Database -> orcl -> HR -> Packages** and then right click on the **P_BULK_BIND** package body and select **Open** to view the PL/SQL code.



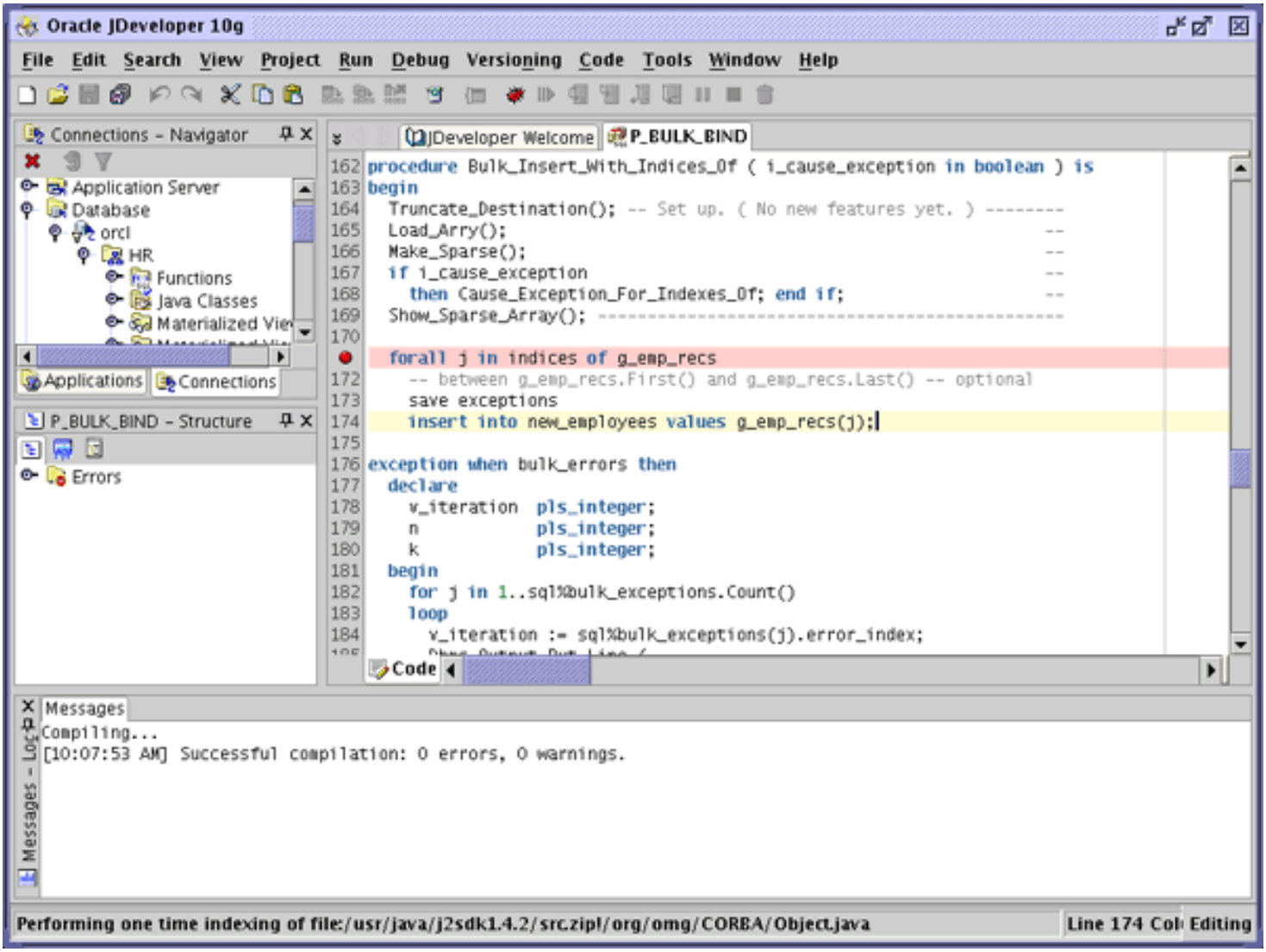
☐ **2.** You need to compile this package body. Right click on the **P_BULK_BIND** package body again and select **Make P_BULK_BIND.**



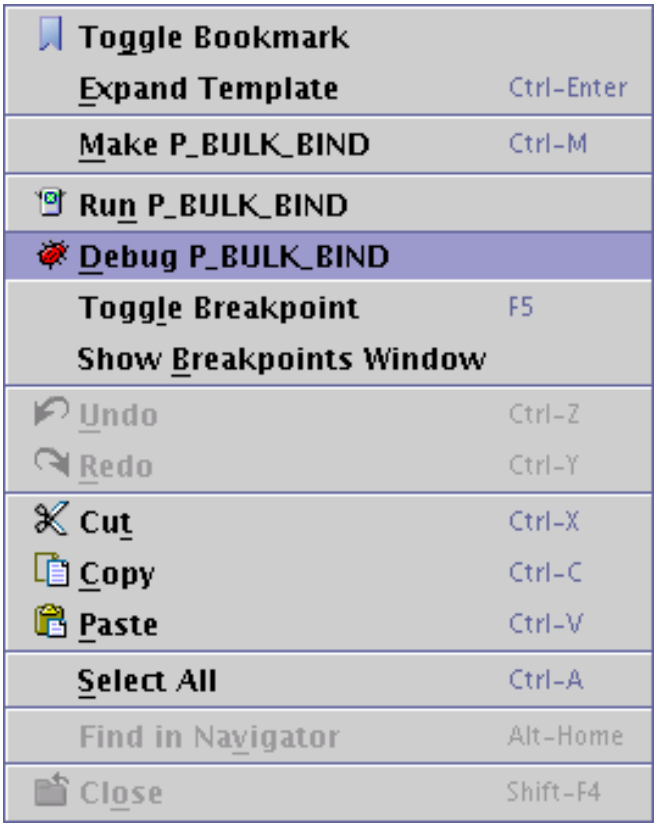
☐ **3.** Right click in the Code Editor gutter and select **Toggle Line Numbers** .



☐ **4.** To set the breakpoint, scroll down to the statement **`forall j in indices of g_emp_recs`** (approximately at line 171) and click on the line number.



☐ **5.** To start debugging, right click in the Code Editor window and select **Debug P_BULK_BIND** .



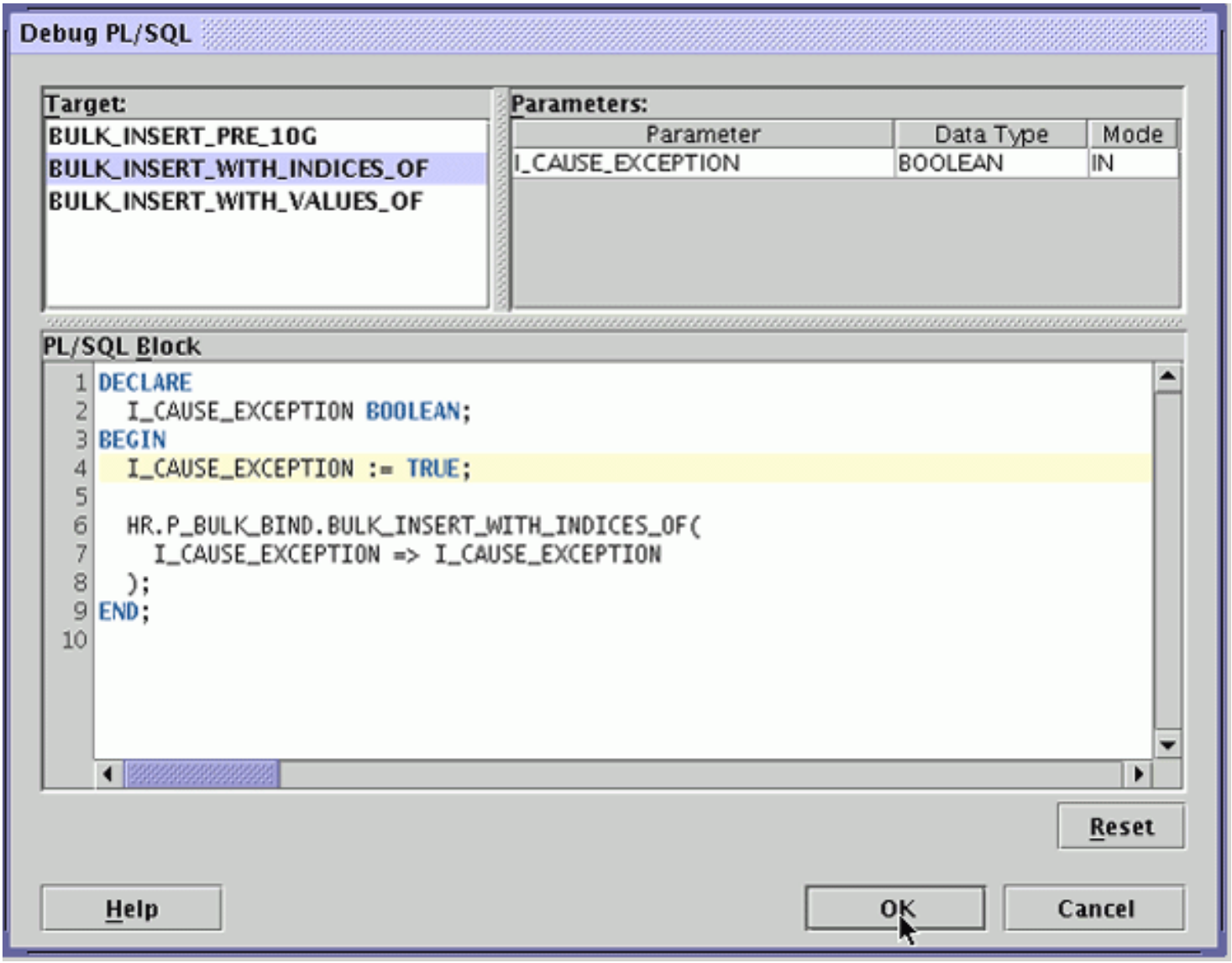
☐ **6.** This will bring up the Debug PL/SQL window, which displays the three global procedures within the package. Select the procedure **`Bulk_Insert_With_Indices_Of`** . This procedure takes one input parameter, that is a Boolean flag called " `I_CAUSE_EXCEPTION` ". The value for this parameter is to be set to true. In the PL/SQL block area, change

```
I_CAUSE_EXCEPTION := NULL;
```

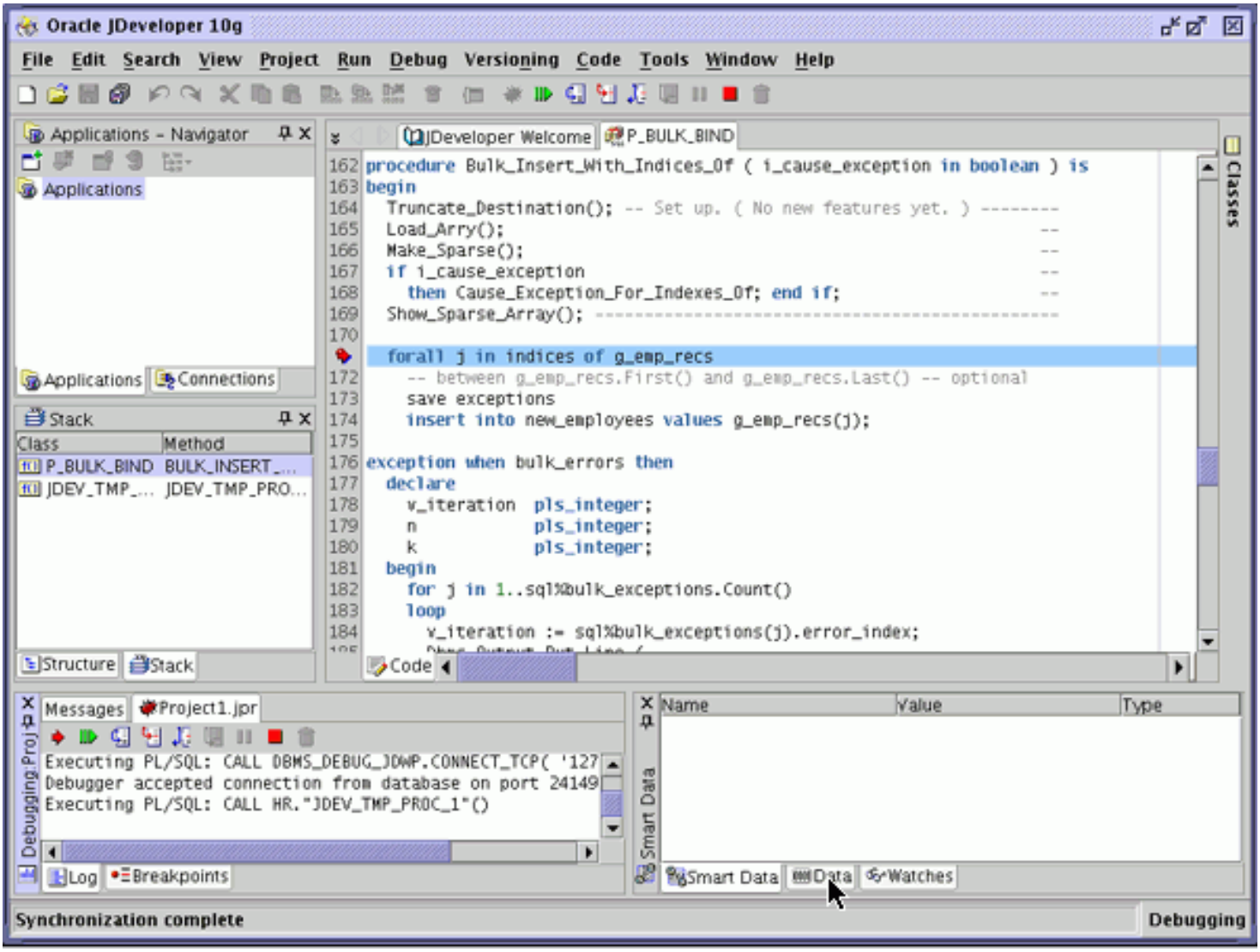
to

```
I_CAUSE_EXCEPTION := TRUE;
```

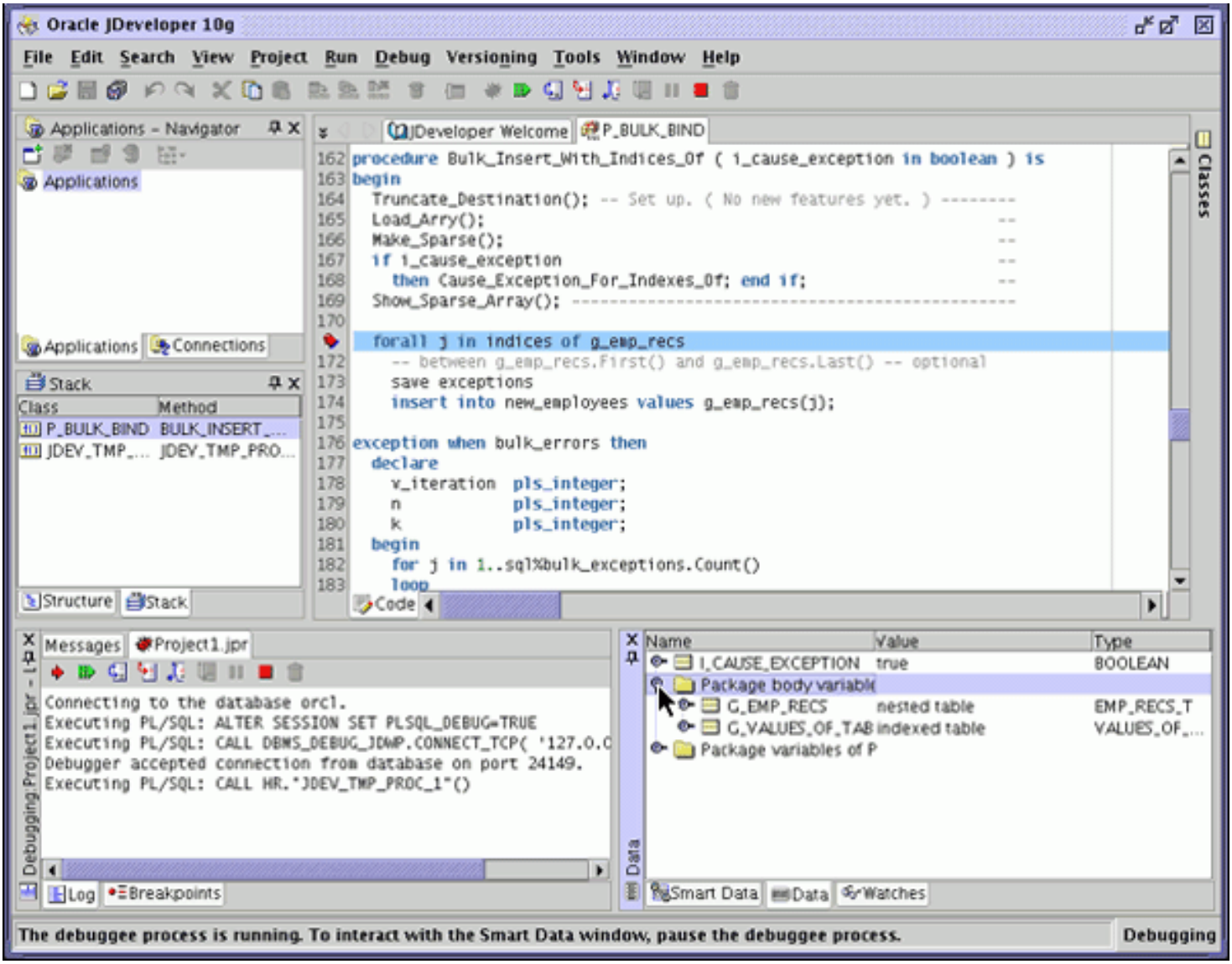
Then click **OK** .



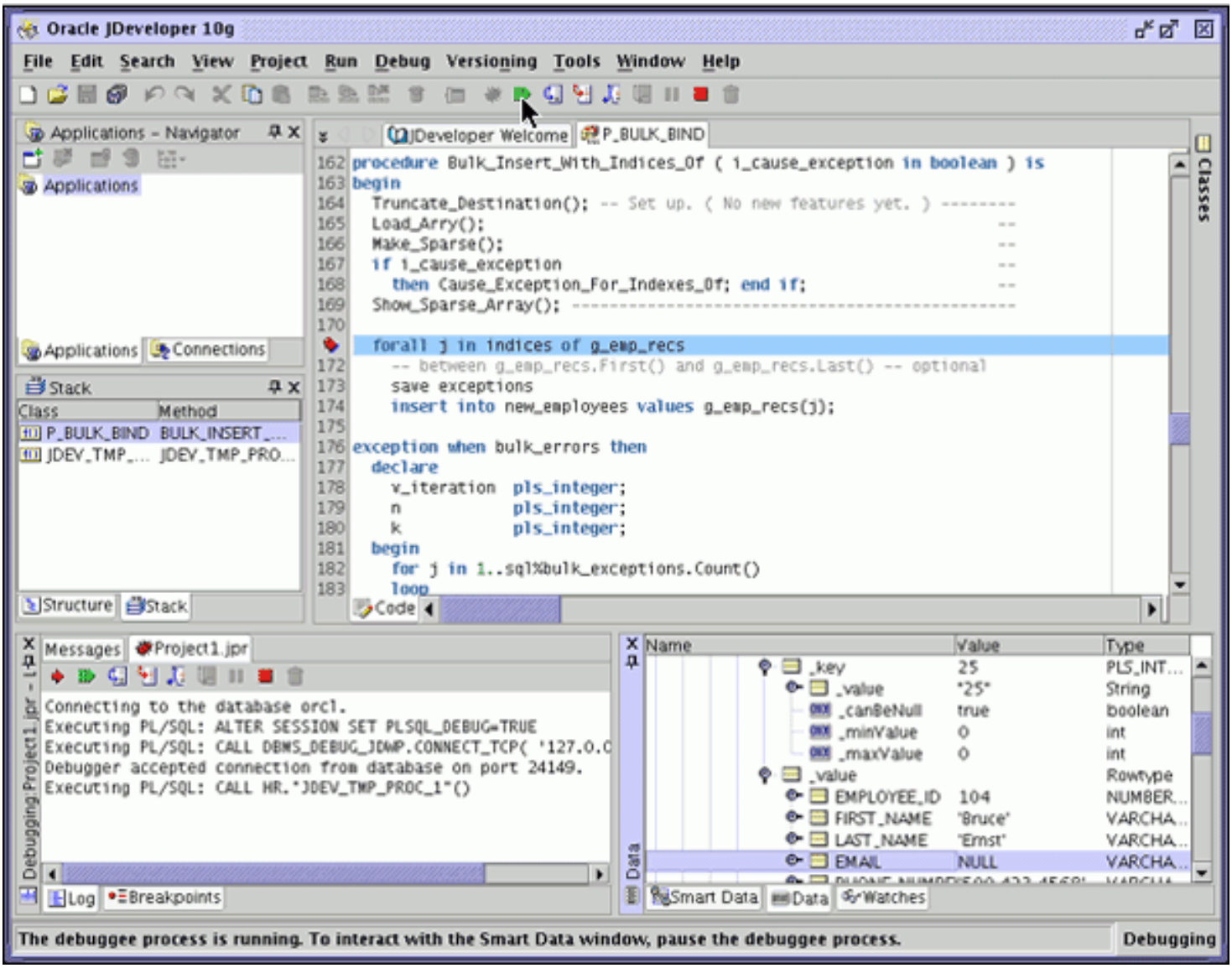
☐ **7.** Upon continuing, the debugger executes the statements up to the first breakpoint. At this point, the breakpoint statement (to be executed) next is highlighted. The Stack window displays the procedure being executed. Click the **Data** tab to display the variables and their values, within the current procedure.



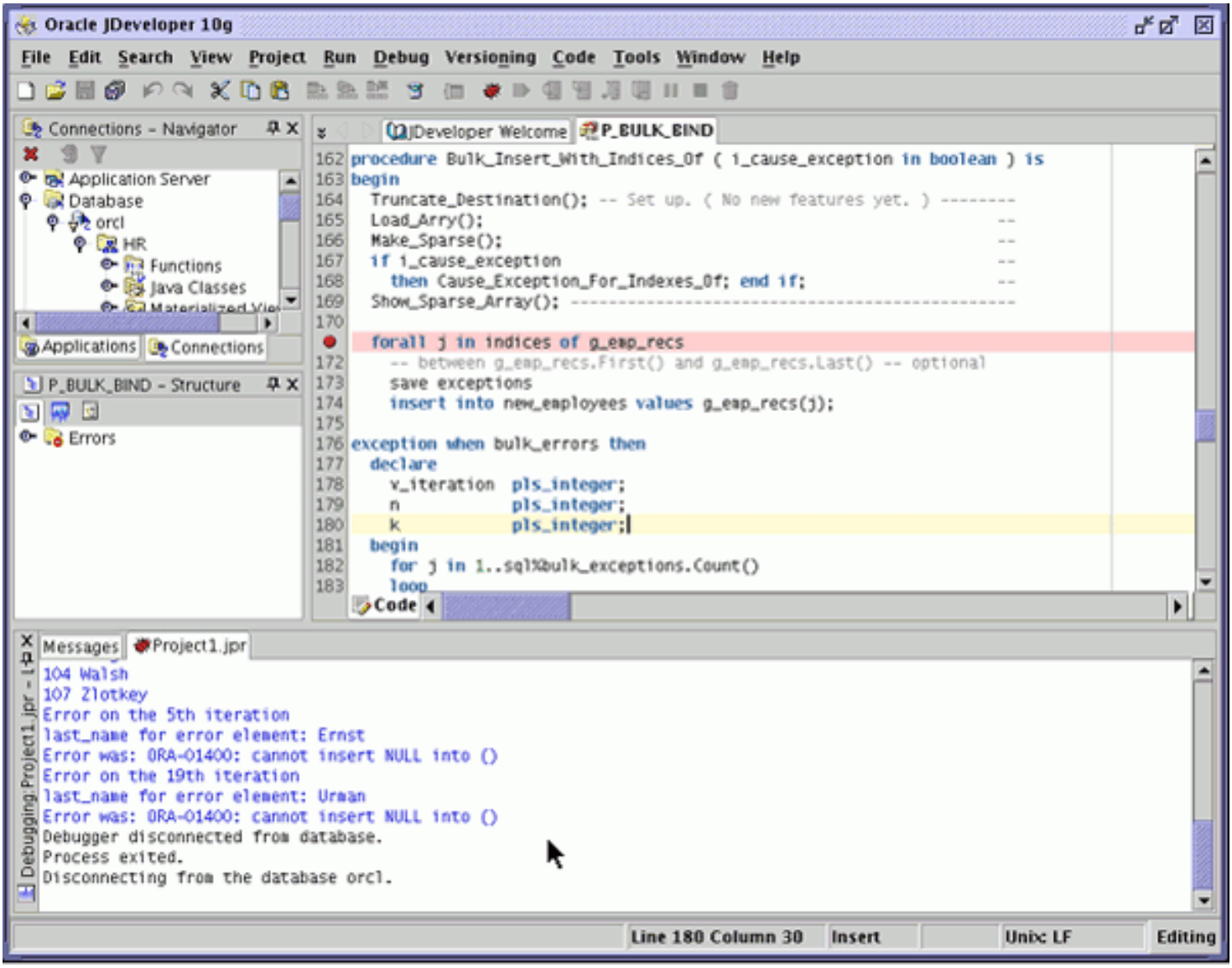
☐ **8.** Expand **Package body variables of P_BULK_BIND** . You see two variables - `G_EMP_RECS` and `G_VALUES_OF_TAB` . `G_EMP_RECS` is the PL/SQL table used by the application to bulk insert data into the `NEW_EMPLOYEES` table.



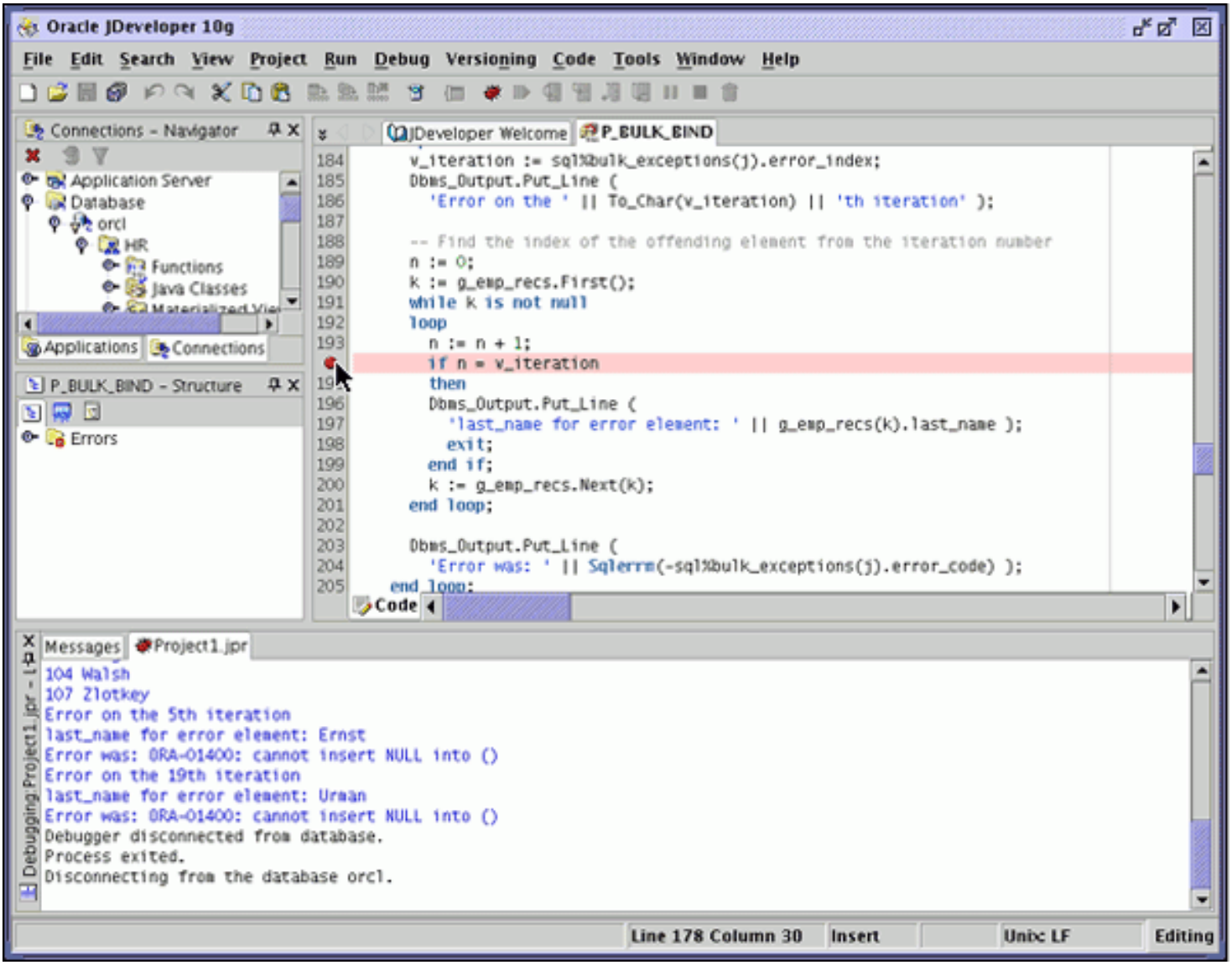
☐ **9.** Expand **`G_EMP_RECS -> _values -> [25] -> _value`** . Notice that `_value` shows the actual record values and the `EMAIL` column is null for this record. To observe the effect, click on the **Resume** button (green button in Main Toolbar).



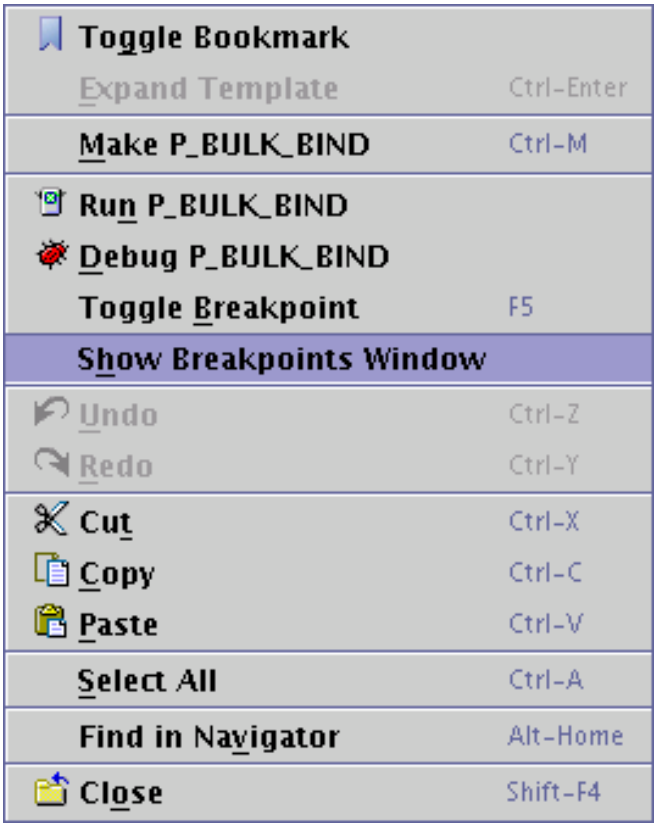
☐ **10.** The procedure is now executed to the end and the "Log" window displays program output at the end of which are the error messages for the 2 records for which insert failed.



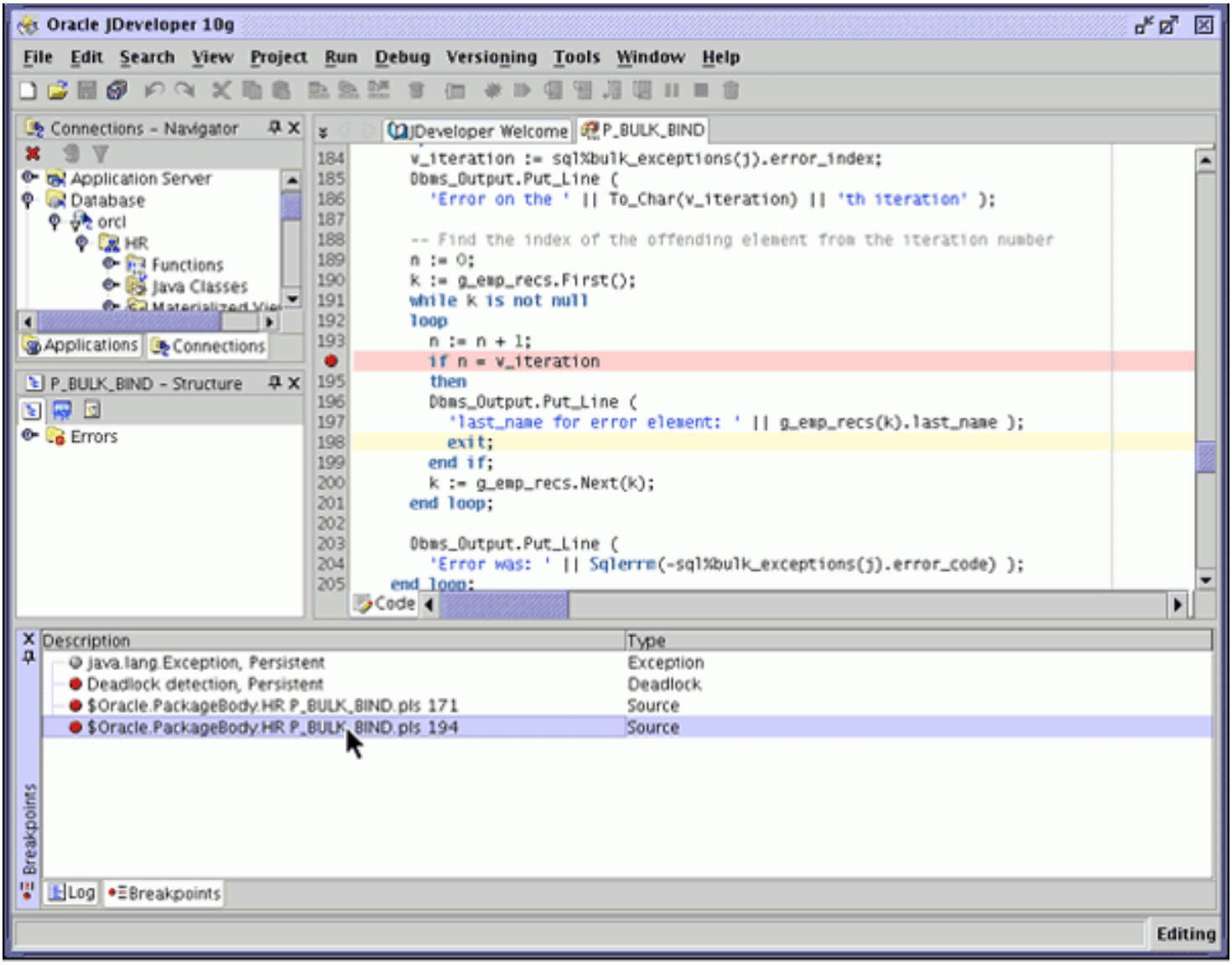
☐ **11.** To debug the problem, you can start the debugger with a conditional breakpoint. The goal is to halt execution in the exception block for the erroneous record. For this, set a breakpoint at the line `if n = v_iteration` . Click the line number **194** .



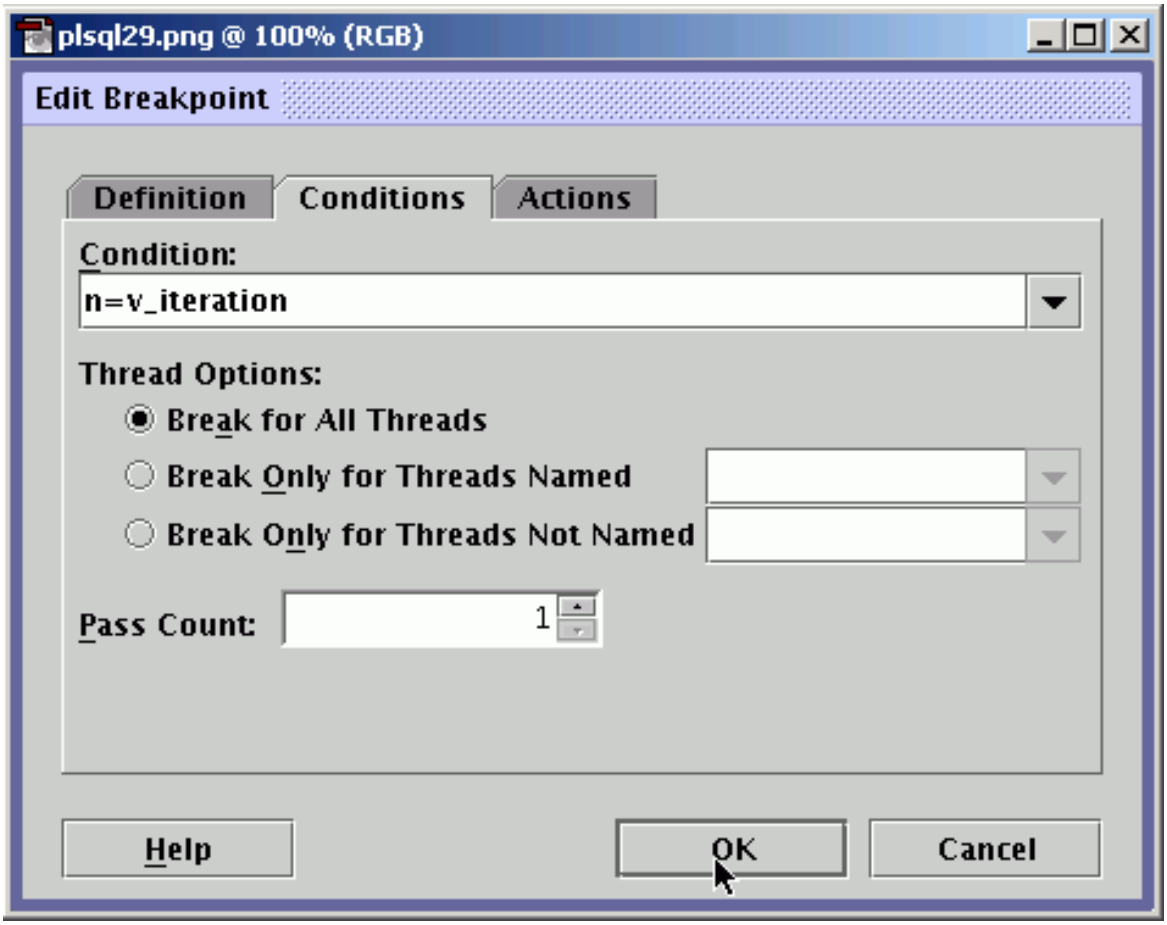
12. ☐ Right click and select **Show BreakPoints Window** .



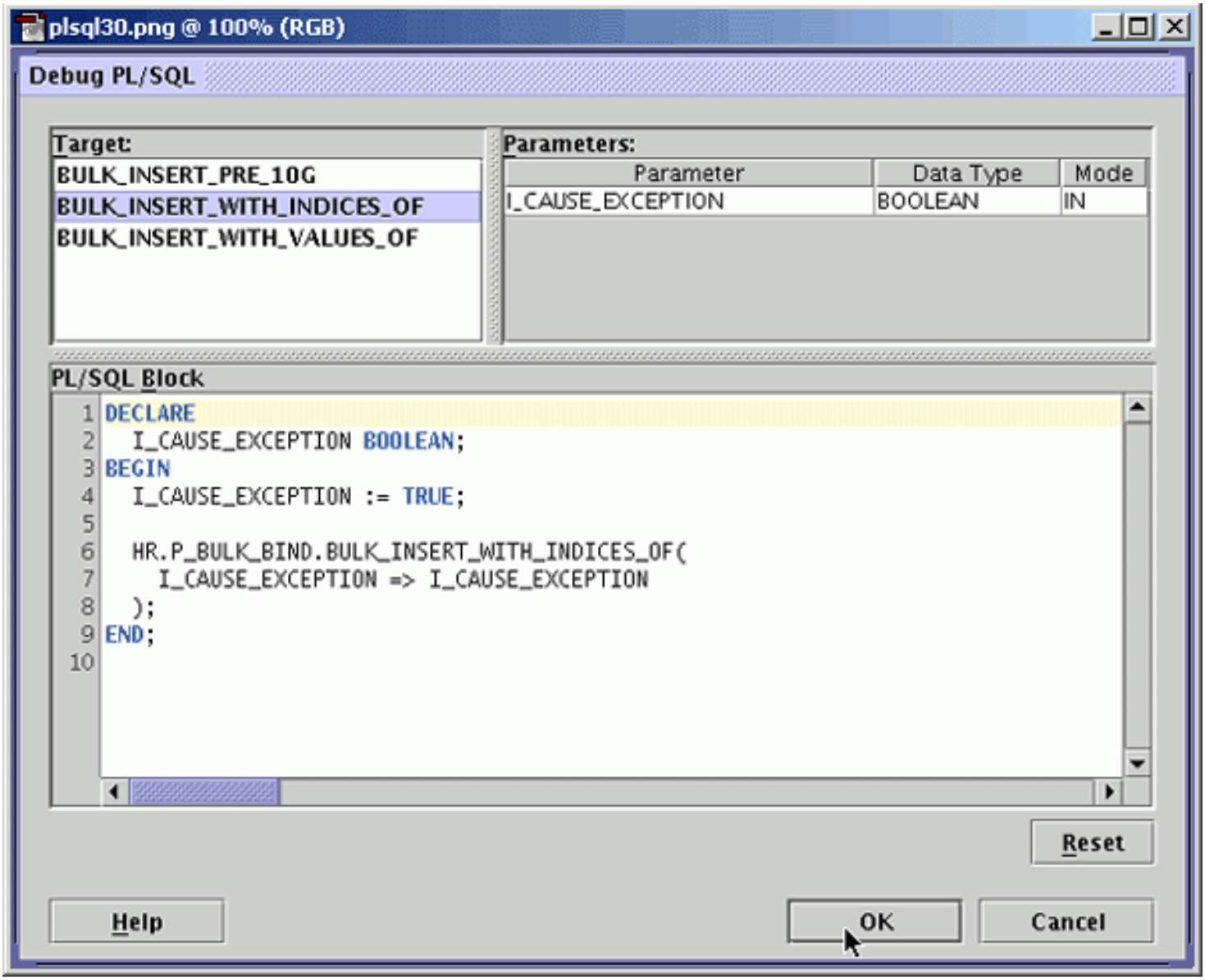
13. ☐ Right click on the breakpoint **`$Oracle.PackageBody.HR.P_BULK_BIND.pls 194`** , right click and select **Edit** .



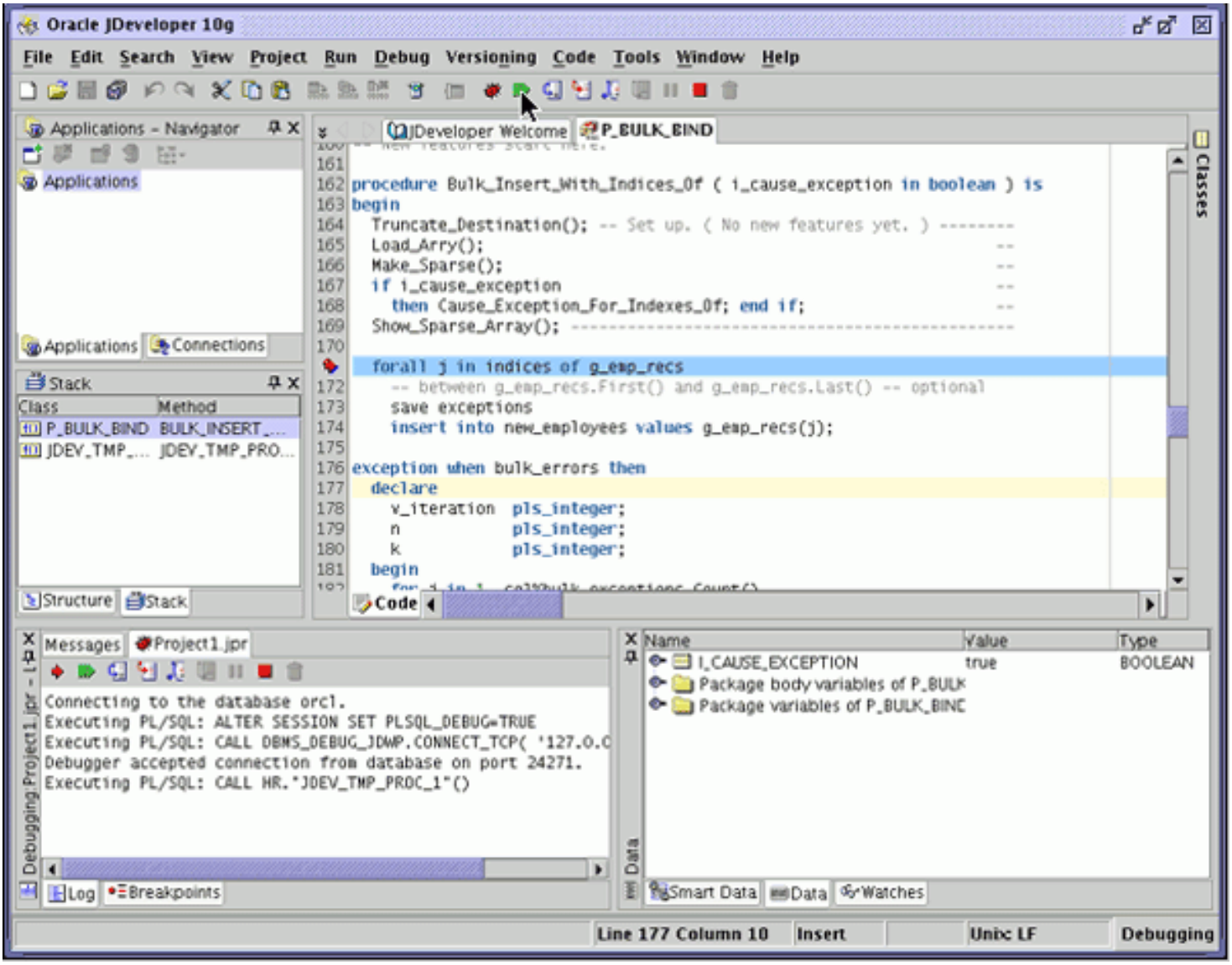
☐ **14.** Select the **Conditions** tab. In the Condition: field enter the condition **`n = v_iteration`** . Only when this condition is evaluated to true, the debugger stops at the breakpoint. Then click **OK** .



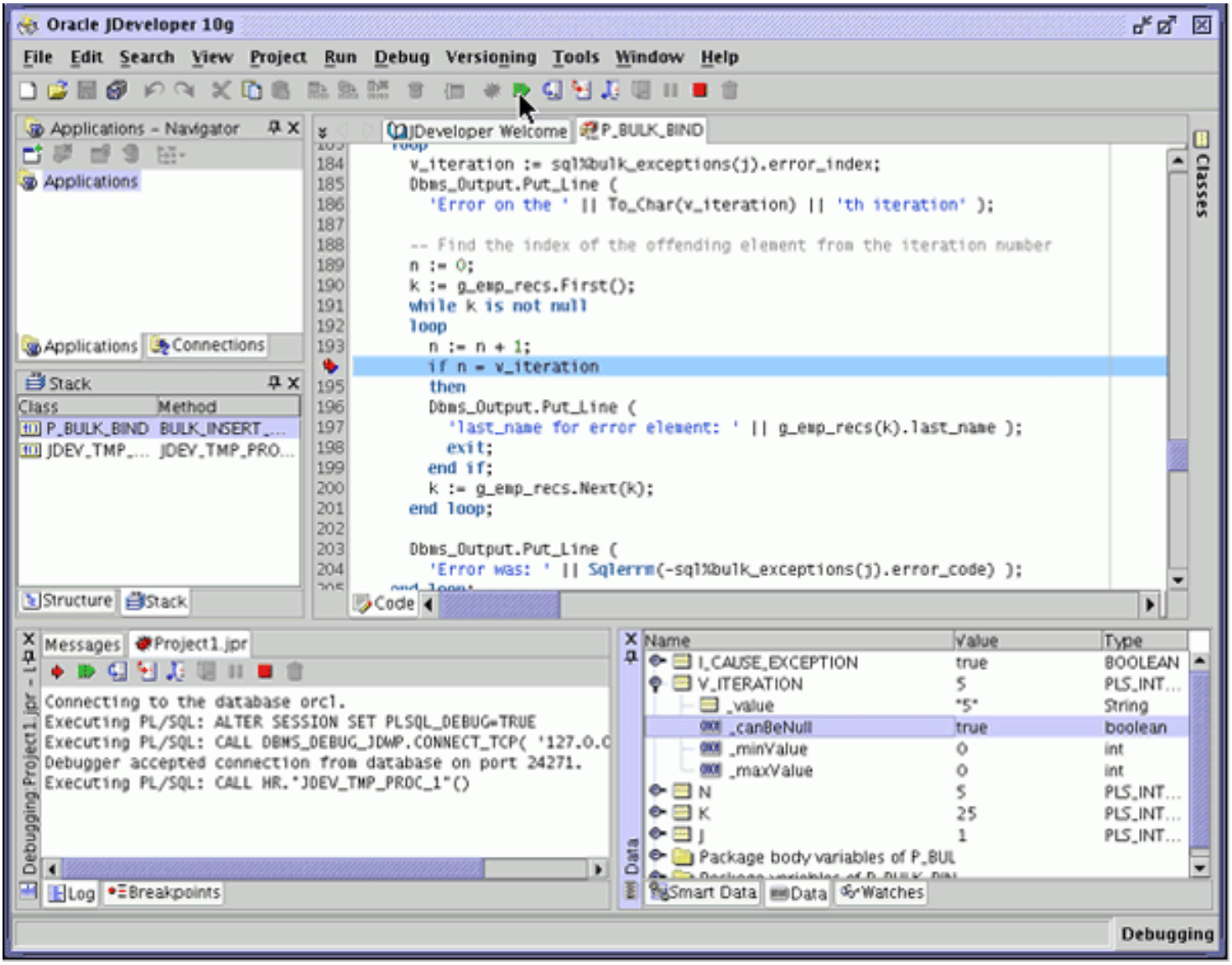
☐ **15.** Start the debugger again. Right click in the Code Editor window and select **Debug P_BULK_BIND** . At the Debug PL/SQL window, select the procedure **`Bulk_Insert_With_Indices_Of.`** `I_CAUSE_EXCEPTION` should still be set to **TRUE** . Click **OK** .



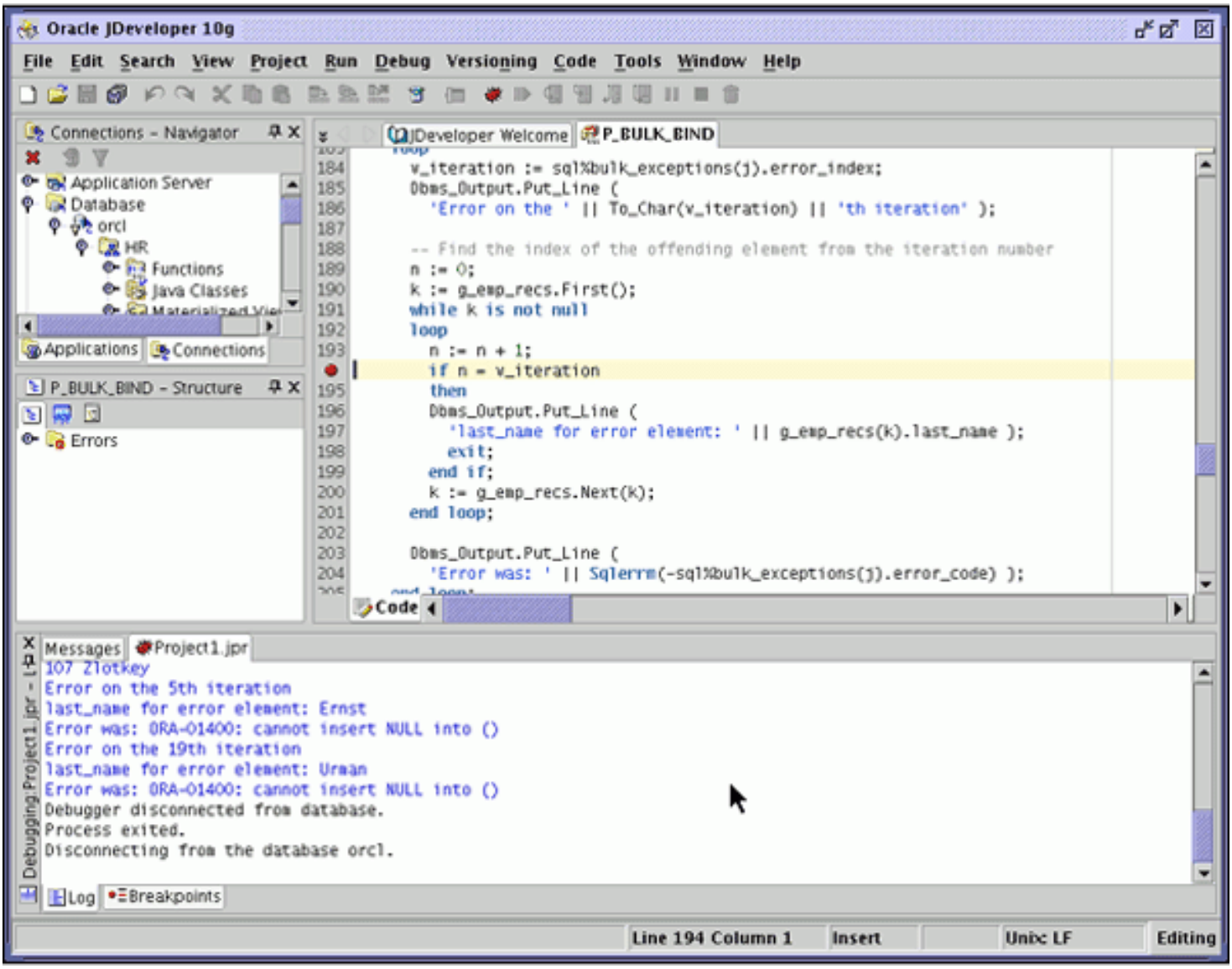
☐ **16.** The execution halts when the erroneous record is reached (while processing the exceptions). Click **Resume** .



☐ **17.** The second breakpoint is reached. The exception is raised for the 5th iteration, which is record index [25]. Upon expanding the record, the 'email' value for this record is null. Click **Resume** .



18. ☐ The results are displayed.


Oracle JDeveloper 10g
File Edit Search View Project Run Debug Versioning Code Tools Window Help
Connections - Navigator
Application Server
Database
orcl
HR
Functions
Java Classes
Applications
Connections
P_BULK_BIND - Structure
Errors
JDeveloper Welcome
P_BULK_BIND
v_iteration := sql%bulk_exceptions(j).error_index;
Dbms_Output.Put_Line (
'Error on the ' || To_Char(v_iteration) || 'th iteration');
-- Find the index of the offending element from the iteration number
n := 0;
k := g_emp_recs.First();
while k is not null
loop
n := n + 1;
if n = v_iteration
then
Dbms_Output.Put_Line (
'last_name for error element: ' || g_emp_recs(k).last_name);
exit;
end if;
k := g_emp_recs.Next(k);
end loop;
Dbms_Output.Put_Line (
'Error was: ' || Sqlerrm(-sql%bulk_exceptions(j).error_code));
Code
Messages
Project1.jpr
107 Zlotkey
Error on the 5th iteration
last_name for error element: Ernst
Error was: ORA-01400: cannot insert NULL into ()
Error on the 19th iteration
last_name for error element: Urman
Error was: ORA-01400: cannot insert NULL into ()
Debugger disconnected from database.
Process exited.
Disconnecting from the database orcl.
Debugging-Project1.jpr
Log
Breakpoints
Line 194 Column 1
Insert
Unix LF
Editing