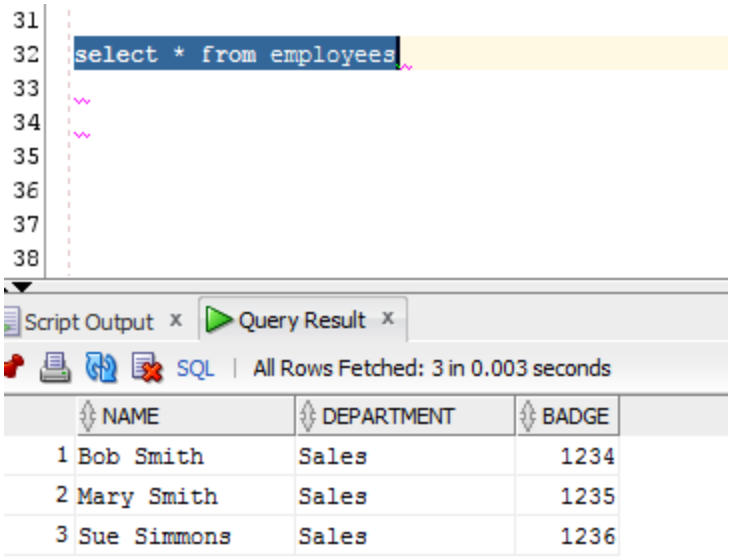
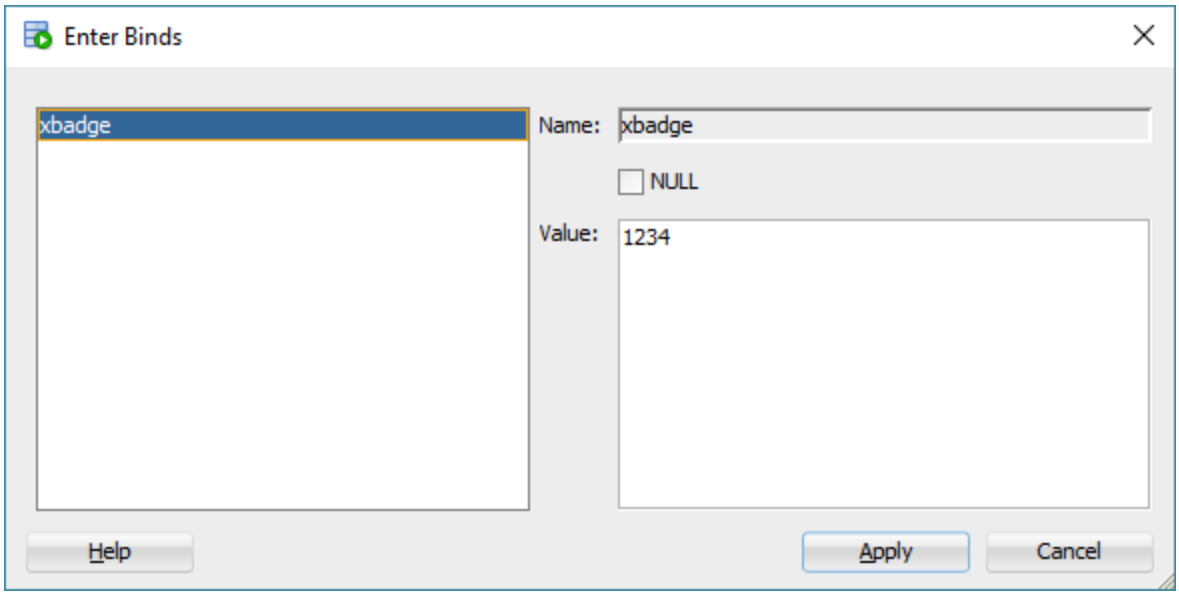
Examples executing within SQL Developer:



Using : for a bind variable

```
DECLARE
   xname employees.name%TYPE;
BEGIN
   SELECT name
   INTO   xname
   FROM   employees
   WHERE  badge = :xbadge;

   DBMS_OUTPUT.PUT_LINE('located row: ' || xname);
END;
/
```



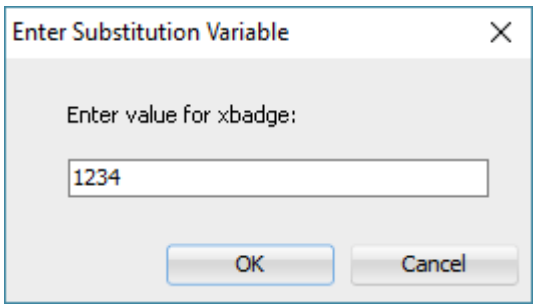
located row: Bob Smith

PL/SQL procedure successfully completed.

Using & for a substitution variable

```
DECLARE
   xname employees.name%TYPE;
BEGIN
   SELECT name
   INTO   xname
   FROM   employees
   WHERE  badge = &xbadge;

   DBMS_OUTPUT.PUT_LINE('located row: ' || xname);
END;
```



located row: Bob Smith

PL/SQL procedure successfully completed.

Here's the version that uses a **bind variable** (`:xbadge`) instead of a substitution variable. This is the cleanest way to prompt once and then use the value inside PL/SQL.

Step 1: Declare the bind variable

In SQL*Plus or SQL Developer:

```
VARIABLE xbadge NUMBER;
```

Step 2: Assign a value to the bind variable

Oracle will prompt you for the value:

```
EXEC :xbadge := &prompt_for_badge;
```

This uses `&` **once** to prompt, then stores the value in `:xbadge`.

Step 3: PL/SQL block using the bind variable

```
DECLARE
    xname employees.name%TYPE;
BEGIN
    SELECT name
    INTO   xname
    FROM   employees
    WHERE  badge = :xbadge;

    DBMS_OUTPUT.PUT_LINE('located row: ' || xname);
END;
/
```

What's happening

- `&prompt_for_badge` prompts the user **once**.
- The value is stored in the bind variable `:xbadge`.
- The PL/SQL block uses `:xbadge` safely and efficiently.
- Bind variables are better for performance and SQL injection protection.

A few things can go wrong with that block depending on how and where you run it. The code *looks* close, but the issue is almost always the same:

Inside a PL/SQL block, `&xbadge` is **text-substituted before compilation**. That means Oracle literally replaces `&xbadge` with whatever you type.

So if you enter:

```
Enter value for xbadge: 123
```

Oracle rewrites your block as:

```
WHERE badge = 123;
```

That part is fine — **but only if**:

- You are running this in SQL*Plus or SQL Developer (which support `&` substitution)
- The value you enter is valid SQL syntax
- The column `badge` exists in `employees`

If any of those conditions fail, you get an error.

The *real* issue most people hit

You never declared a PL/SQL variable named `xbadge`.

So if you *expected* this to work:

```
WHERE badge = xbadge;
```

…it won't, because `xbadge` does not exist in the block.

`&xbadge` is **not** a PL/SQL variable — it is a SQL*Plus text prompt.

Corrected version using substitution variable

If your goal is simply to prompt the user:

```
DECLARE
    xname employees.name%TYPE;
BEGIN
    SELECT name
    INTO   xname
    FROM   employees
    WHERE  badge = &xbadge;

    DBMS_OUTPUT.PUT_LINE('located row: ' || xname);
END;
/
```

This works **as long as**:

- You run it in SQL*Plus or SQL Developer
- You enter a valid number when prompted

Corrected version using a real PL/SQL variable

If you want a proper PL/SQL variable, do this:

```
DECLARE
    xname  employees.name%TYPE;
    xbadge employees.badge%TYPE := &xbadge;
BEGIN
    SELECT name
    INTO   xname
    FROM   employees
    WHERE  badge = xbadge;

    DBMS_OUTPUT.PUT_LINE('located row: ' || xname);
END;
/
```

Now:

- `&xbadge` prompts the user
- The value is stored in `xbadge`
- The `WHERE` clause uses a real PL/SQL variable

Here's the clean, correct version that uses a **bind variable** (`:xbadge`) instead of a substitution variable. This is the preferred approach when you want PL/SQL to use a value supplied at runtime without doing text substitution.

Step 1: Declare the bind variable

Run this in SQL*Plus or SQL Developer:

sql
```
VARIABLE xbadge NUMBER;
```

Step 2: Assign a value to the bind variable

Oracle will prompt you for the value using `&` just once:

```
EXEC :xbadge := &prompt_for_badge;
```

After this, `:xbadge` holds the value and can be reused without further prompts.

Step 3: PL/SQL block using the bind variable

```
DECLARE
```

```
    xname employees.name%TYPE;
BEGIN
    SELECT name
    INTO   xname
    FROM   employees
    WHERE  badge = :xbadge;

    DBMS_OUTPUT.PUT_LINE('located row: ' || xname);
END;
/
```

Why this version is better

- `:xbadge` is a **real bind variable**, not a text substitution.
- The value is stored once and reused safely.
- Bind variables improve performance and prevent SQL injection.
- The PL/SQL block is cleaner and doesn't rely on `&` inside the code.

using **ACCEPT** is a classic SQL*Plus / SQL Developer approach when you want a clean prompt *outside* the PL/SQL block, without any `&` inside the code.

Here's the full, correct pattern.

Step 1: Prompt the user with `ACCEPT`

```
ACCEPT xbadge PROMPT 'Enter badge number: '
```

This stores the user's input in a SQL*Plus substitution variable named `xbadge`.

Step 2: Use the accepted value inside PL/SQL

```
DECLARE
    xname employees.name%TYPE;
BEGIN
    SELECT name
    INTO   xname
    FROM   employees
    WHERE  badge = &xbadge;

    DBMS_OUTPUT.PUT_LINE('located row: ' || xname);
END;
/
```

What's happening

- `ACCEPT` prompts the user **before** the PL/SQL block runs.
- The value is stored in the substitution variable `&xbadge`.
- The PL/SQL block uses that value without prompting again.
- No bind variables needed unless you want them.

Optional: Convert the accepted value into a real PL/SQL variable

If you prefer to avoid `&` inside the block:

```
+++++++++++++
ACCEPT xbadge PROMPT 'Enter badge number: '

DECLARE
    xname  employees.name%TYPE;
    vbadge employees.badge%TYPE := &xbadge;
BEGIN
    SELECT name
    INTO   xname
    FROM   employees
    WHERE  badge = vbadge;

    DBMS_OUTPUT.PUT_LINE('located row: ' || xname);
END;
/
```

This is the cleanest version if you want the PL/SQL block to use only real variables.

Here's the **cleanest and most professional** way to combine `ACCEPT` with a PL/SQL block, using **no ampersands inside the block** and **no bind variables**. This is the pattern people use in real SQL*Plus and SQL Developer scripts.

Step 1: Prompt the user with `ACCEPT`

```
ACCEPT xbadge PROMPT 'Enter badge number: '
```

This pauses and waits for the user to type a badge number. The value is stored in the SQL*Plus substitution variable `&xbadge`.

Step 2: Use the accepted value inside PL/SQL

Version A — Use the substitution variable directly

```
DECLARE
    xname employees.name%TYPE;
BEGIN
    SELECT name
    INTO   xname
    FROM   employees
    WHERE  badge = &xbadge;

    DBMS_OUTPUT.PUT_LINE('located row: ' || xname);
END;
/
```

This works fine, but many developers prefer to avoid `&` inside PL/SQL.

Step 3: Cleaner version — move the value into a PL/SQL variable

This avoids substitution inside the SQL statement.

```
ACCEPT xbadge PROMPT 'Enter badge number: '

DECLARE
    vbadge employees.badge%TYPE := &xbadge;
```

```
    xname  employees.name%TYPE;
BEGIN
    SELECT name
    INTO   xname
    FROM   employees
    WHERE  badge = vbadge;

    DBMS_OUTPUT.PUT_LINE('located row: ' || xname);
END;
/
```

Why this version is cleaner

- The PL/SQL block contains **only PL/SQL variables**, no substitution variables.
- `ACCEPT` handles all user interaction up front.
- The block is reusable and easier to maintain.