Database Management Systems

Lab 6

Oracle PL/SQL and Stored Procedures

Lab Topics

☐ **Oracle PL/SQL** basics.

☐ Using cursors in **Oracle PL/SQL** procedures/functions.

☐ Creating **Oracle PL/SQL** stored procedures/functions.

PL/SQL = Procedural Language/SQL, 即过程化SQL语言

Oracle PL/SQL (1)

☐ **Oracle PL/SQL** (**Procedural Language/SQL**) allows **SQL** statements to be embedded into a procedural programming language.

☐ **Oracle PL/SQL** extends the capabilities of **SQL** by adding functionality that is supported by procedural programming languages.

☐ An **Oracle PL/SQL** can be

■ a procedure, which does not return a value.

■ a function, which returns a value using the **return** keyword.

Oracle PL/SQL (2)

- **Oracle PL/SQL** statements:
  - are *case insensitive*.
  - use C style comments /*...*/.
  - use := operator to assign values to a variable.
  - use = operator for comparison.

- A block is the basic processing unit in **Oracle PL/SQL** and is delimited by **begin**...**end**.

- <u>ALLOWED</u> **SQL** statements: **select**, **insert**, **update** and **delete** (i.e., DML statements)

- <u>NOT ALLOWED</u> **SQL** statements: **create**, **drop**, **alter**, **rename** (i.e., DDL statements).

Oracle PL/SQL Basic Structure

create or replace procedure *procedure_name* [**as** | **is**]

      Declarative section: declaration of variables, types, and local subprograms go here.

begin   **Executable section:** procedural and **SQL** statements go here. This is the only required section of the block.

exception **Exception handling section:** error handling statements go here.

end;

Example:

```
create or replace procedure IncrementRoomNumber as
    room    Department.roomNo%type; -- room is of type roomNo
begin
    select roomNo into room from Department where departmentId='COMP';
    if (room>3000 and room<4000) then
        update Department set roomNo=room+1000 where departmentId='COMP';
    else
        update Department set roomNo=5528 where departmentId='COMP';
    end if;
end;
```

Declaring Variables

☐ A variable's data type can be either

- ■ an SQL basic type (i.e., **number**, **int**, **char**, **varchar2**, **date**, etc.) or
- ■ the same type as a table attribute (*table_name.attribute_name%***type**) or a table row (*table_name%***rowtype**).

Examples

- ■ Declare count to be a variable of basic type number.

  count **number**;

- ■ Declare projectors to be a variable with the same type as the numberProjectors attribute in the Facility table.

  projectors Facility.numberProjectors%**type**;

- ■ Declare facilityRecord to be a variable that is the same type as a row (tuple).

  facilityRecord Facility%**rowtype**;

Select-Into Statement

> The **select-into** statement must retrieve *at most* one record from only one table (i.e., cannot do a join with the **select-into** statement).

Example: Extract data of the Math department from the Department table into a table called MathDept.

```
create or replace procedure ExtractMathRecords as
    -- deptName is the same type as departmentName in the Department table
    deptName    Department.departmentName%type;
    -- deptRoom is the same type as roomNo in the Department table
    deptRoom    Department.roomNo%type;
begin
    select departmentName, roomNo into deptName, deptRoom
        from Department where departmentId='MATH';
    insert into MathDept values (deptName, deptRoom);
end;
```

Must fetch at most one record

- The value of the attributes departmentName and roomNo in the Department table are assigned to, respectively, the PL/SQL variables deptName and deptRoom by the select statement.

Flow of Control Statements

Sequential control

goto
- Branch to a label unconditionally.

null
- Pass control to the next statement.

return
- Return control to the calling block and possibly return a value (for a function).

Conditional control

if-then, **if-then-else**, **if-then-elsif**
- Conditional processing.

case
- Selects one sequence of statements to execute.

Iterative control

loop *statements* **end loop**;

while *condition* **loop** *statements* **end loop**;
- Executes the loop while *condition* is true.

for *index* **in** [**reverse**] *lower_bound..upper_bound* **loop** *statements* **end loop**;
- Iterates over a range of integers starting either from *lower_bound* to *upper_bound* or in *reverse* order.

exit / **exit when** *condition*
- Exits current loop completely either unconditionally or when *condition* is true.

continue / **continue when** *condition*
- Exits current loop iteration and continues with the next iteration either unconditionally or when *condition* is true.

If-Then-Else Example (1)

```
create or replace procedure IncrementRoomNumber as
    room    Department.roomNo%type; -- room is of type roomNo
begin
    select roomNo into room from Department where departmentId='COMP';
    if (room>3000 and room<4000) then
        update Department set roomNo=room+1000 where departmentId='COMP';
    else
        update Department set roomNo=5528 where departmentId='COMP';
    end if;
end;
```

This procedure adds 1000 to the room number of the COMP department if the room starts with 3. The room number is set to 5528 if the room number does not start with 3.

If-Then-Else Example (2)

```
create or replace procedure IncrementRoomNumber as
    room Department.roomNo%type; -- room is of type roomNo
begin
    -- Incorrect use of select statement
⇒   room := select roomNo from Department where departmentId='COMP';
    if (room>3000 and room<4000) then
        update Department set roomNo=room+1000 where departmentId='COMP';
    else
        update Department set roomNo=5528 where departmentId='COMP';
    end if;
end;
```

Cannot assign the result of a **select** statement to a variable as in the highlighted line since the result of a **select** statement is <u>always</u> a table even if the table contains only one value.

Loop Example

☐ Insert values 1 to 10 into table Testloop.

```
create or replace procedure LoopTest as
    i    Testloop.testValue%type := 1; -- i is of type testValue and is initialized to 1
begin
    loop
        insert into Testloop values (i);
        i := i + 1;
        exit when i > 10;
    end loop;
end;
```

Note: A **loop** can be terminated by the **exit** or **exit when** keywords.

Loop Label Example

☐ A loop label appears at the beginning of a loop statement enclosed by double angle brackets.

```
create or replace procedure LoopTest as
    i    Testloop.testValue%type := 1; -- i is of type testValue and is initialized to 1
begin
    «myLoop»
    loop
        insert into Testloop values (i);
        i := i + 1;
        exit myLoop when i > 10;
    end loop;
end;
```

Note: Several levels of nested loops can be terminated using a loop label.

For-Loop Example

☐ Increase the number of projectors in a department based on a loop counter i.

```
create or replace procedure IncrementProjectors as
    i    number(2) := 1;
begin
    for var in (select * from Facility order by departmentId) loop
        update Facility set numberComputers = numberComputers + 1
            where departmentId=var.departmentId;
        i := i + 1;
    end loop;
end;
```

Note: In this example the **for-loop** acts like a **for-each-loop** where var is assigned the next record in the result of the **select** statement in each iteration of the **for-loop**.

Cursors(光标)

☐ If a **select** statement returns more than one record, a cursor is needed to process the result records one-at-a-time.

☐ A cursor is like a pointer that points to a single record in the result of a **select** statement.

☐ When used with the **for-loop** statement, the cursor iterates over the result records one-at-a-time allowing the values in a record to be accessed and manipulated.

☐ A cursor is declared in the **declare** section using the syntax:

　　cursor *cursor_name* **is** *select_statement*;

Example: Declare a cursor that retrieves all the Facility table records.

　　　　cursor facilityCursor **is select** * **from** Facility;

How to Use Cursors

☐ **Explicit cursor**

- Is activated by the **open** command.

- Fetches records one-at-a-time using the **fetch** command.

- The status %notfound returns true when all the records are fetched.

- Needs to be closed with the **close** command so as to free up resources.

☐ **Implicit cursor**

- Is activated using the **for-loop** statement.

- The *cursor_name* replaces the range limit so the loop ranges from the first record of the cursor to the last record of the cursor.

Cursor Status

☐ The possible values of a cursor status are:

- ■ Determine whether the previous fetch failed.

  cursor_name%notfound

- ■ Determine whether the previous fetch succeeded.

  cursor_name%found

- ■ Determine the number of records fetched so far.

  cursor_name%rowcount

- ■ Determine whether the cursor is still open.

  cursor_name%isopen

☐ Cursor status can only be used with an explicit cursor.

Explicit Cursor Example

```
create or replace procedure InsertExample as
    varDeptId      Facility.departmentId%type;
    varComputers   Facility.numberComputers%type;
    cursor facilityCursor is select departmentId, numberComputers from Facility;
begin
    open facilityCursor;
    loop
        fetch facilityCursor into varDeptId, varComputers;
        exit when facilityCursor%notfound;
        insert into ResultTable values (varDeptId, varComputers);
    end loop;
    close facilityCursor;
end;
```

> The facilityCursor retrieves records from the Facility table and inserts the values one-by-one into another table called ResultTable.

Implicit (For-Loop) Cursor Example (1)

```
create or replace procedure InsertExample as
    varDeptId        Facility.departmentId%type;
    varComputers   Facility.numberComputers%type;
    cursor facilityCursor is select departmentId, numberComputers from Facility;
begin
    for record in facilityCursor loop
        varDeptId := record.departmentId;
        varComputers := record.numberComputers;
        insert into ResultTable values (varDeptId, varComputers);
    end loop;
end;
```

facilityCursor defines what data should be retrieved when the select statement is executed.

Each time through the for-loop, the variable record is assigned the next record from the cursor faciltyCursor which contains the result of the select statement.

This is the same example as on the previous slide, but using a for-loop, which automatically opens the cursor, exits the loop when there are no more records in the cursor and closes the cursor.

Implicit (For-Loop) Cursor Example (2)

- ❑ The FacilityCursor on the previous slide is automatically opened by the **for-loop**.

- ❑ The variable record is of data type rowtype, but there is no need to declare it.

- ❑ Code inside the **for-loop** is executed once for each row of the cursor, and each time the two attributes departmentId, and departmentName are copied into record.

- ❑ The data in record can be accessed directly (as shown in the code).

- ❑ The **for-loop** terminates automatically once all the records in the cursor are fetched.

- ❑ The cursor is then closed automatically.

Creating A Procedure/Function (1)

- ☐ In the Connections navigator pane,

  - ■ right-click the Procedures or Functions node;

  - ■ select New Procedure… or New Function… in the context menu as shown to the right.

- ☐ In the Create Procedure dialog, shown on the next slide,

  - ■ enter a name for the procedure (e.g., CourseEnrollmentReport);

  - ■ specify any parameters by clicking the + symbol to add a parameter and specifying a name, mode, datatype and possibly a default value for the parameter;

  - ■ click the OK button.
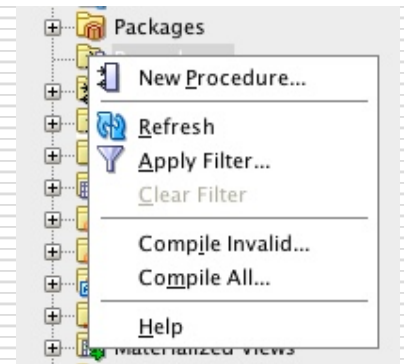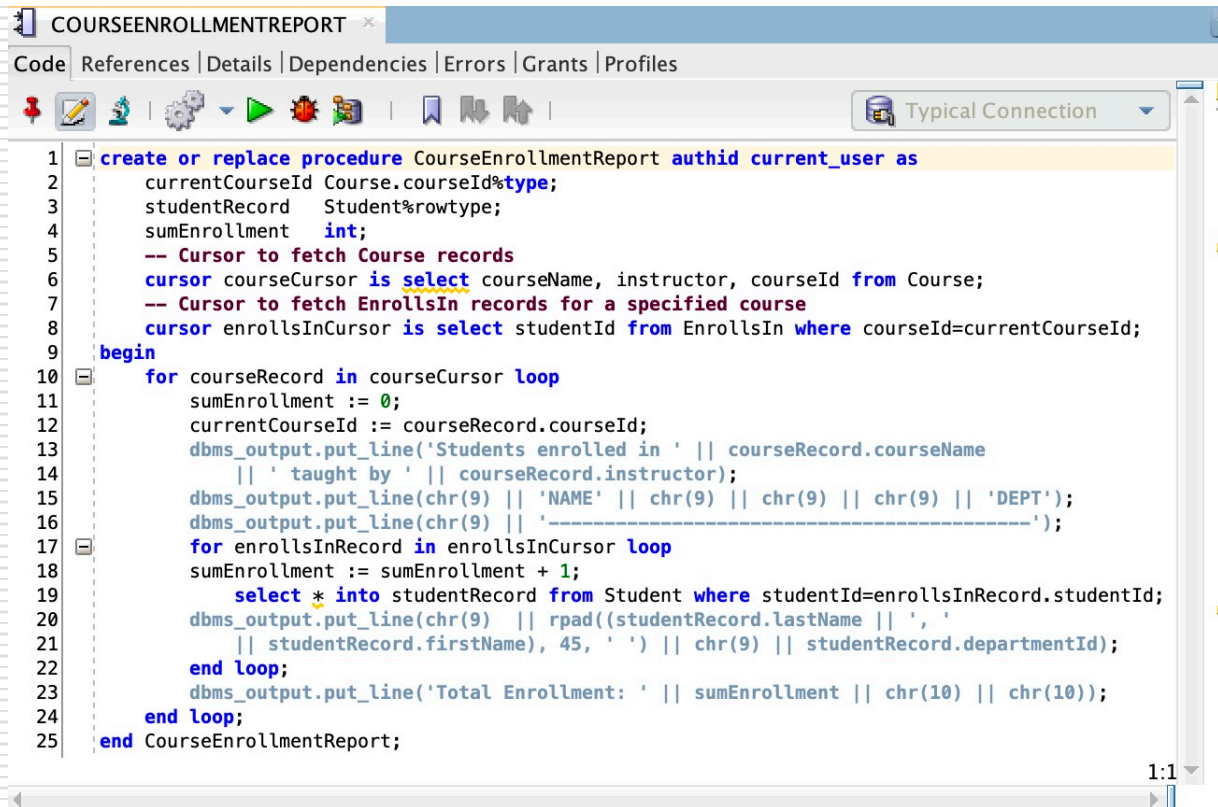
Creating A Procedure/Function (2)

☐ An outline of the procedure with a **NULL** executable section is created as shown in the figure at the bottom right.

☐ Add any declarations before the **BEGIN** statement and executable code between the **BEGIN** and **END** statements as shown on the next slide.
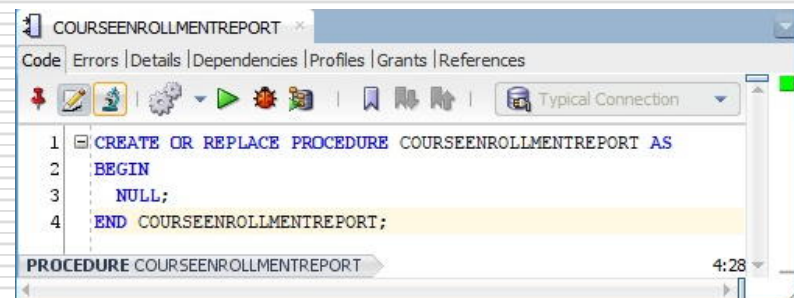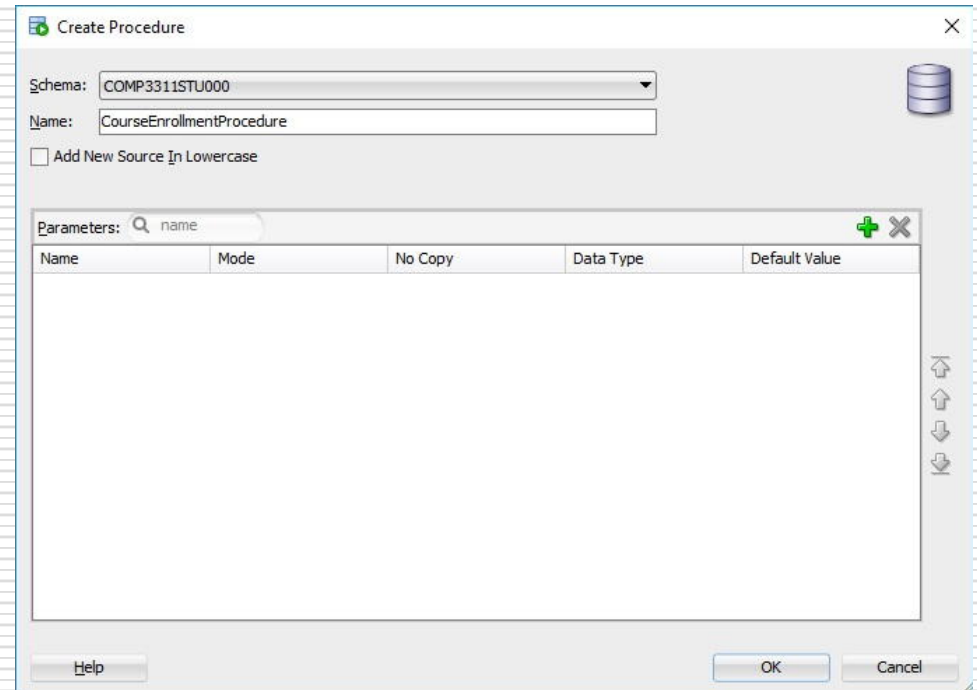
Creating A Procedure/Function (3)


```
  COURSEENROLLMENTREPORT  ×
 Code | References | Details | Dependencies | Errors | Grants | Profiles

                                                          Typical Connection   ▼

 1   ⊟ create or replace procedure CourseEnrollmentReport authid current_user as
 2        currentCourseId Course.courseId%type;
 3        studentRecord    Student%rowtype;
 4        sumEnrollment     int;
 5        -- Cursor to fetch Course records
 6        cursor courseCursor is select courseName, instructor, courseId from Course;
 7        -- Cursor to fetch EnrollsIn records for a specified course
 8        cursor enrollsInCursor is select studentId from EnrollsIn where courseId=currentCourseId;
 9     begin
10   ⊟    for courseRecord in courseCursor loop
11            sumEnrollment := 0;
12            currentCourseId := courseRecord.courseId;
13            dbms_output.put_line('Students enrolled in ' || courseRecord.courseName
14                || ' taught by ' || courseRecord.instructor);
15            dbms_output.put_line(chr(9) || 'NAME' || chr(9) || chr(9) || chr(9) || 'DEPT');
16            dbms_output.put_line(chr(9) || '-----------------------------------------------');
17   ⊟        for enrollsInRecord in enrollsInCursor loop
18            sumEnrollment := sumEnrollment + 1;
19                select * into studentRecord from Student where studentId=enrollsInRecord.studentId;
20            dbms_output.put_line(chr(9)  || rpad((studentRecord.lastName || ', '
21                || studentRecord.firstName), 45, ' ') || chr(9) || studentRecord.departmentId);
22            end loop;
23            dbms_output.put_line('Total Enrollment: ' || sumEnrollment || chr(10) || chr(10));
24        end loop;
25     end CourseEnrollmentReport;
                                                                                  1:1
```

> **IMPORTANT**
> The name of the procedure must match underline exactly
> the name on the final **END** statement.

Creating A Procedure/Function (4)

☐ Sample output of the CourseEnrollmentReport procedure.

```
Students enrolled in Database Management Systems taught by Chen Lei
        NAME                            DEPT
        --------------------------------------------
        Potter, Harry                   COMP
        Grande, Ariana                  COMP
        Greenleaf, Legolas              MATH
        Caesar, Julius                  ELEC
        Jobs, Steve                     COMP
        Gates, Bill                     COMP
        Da Vinci, Leonardo              COMP
        Einstein, Albert                COMP
        Callas, Maria                   COMP
        Musk, Elon                      BUS
        Trump, Donald                   BUS
        Wayne, Bruce                    ELEC
        Buffet, Warren                  BUS
        Newton, Isaac                   MATH
        Turing, Alan                    MATH
        Tesla, Nikola                   ELEC
        Bueller, Ferris                 BUS
        Lazy, Lazzy                     COMP
Total Enrollment: 18

Students enrolled in Internet Computing taught by David Rossiter
        NAME                            DEPT
        --------------------------------------------
        Potter, Harry                   COMP
        Grande, Ariana                  COMP
        Greenleaf, Legolas              MATH
        Clarke, Edith                   ELEC
        Caesar, Julius                  ELEC
        Jobs, Steve                     COMP
        Gates, Bill                     COMP
        Da Vinci, Leonardo              COMP
        Einstein, Albert                COMP
        Callas, Maria                   COMP
        Redford, Robert                 MATH
        Musk, Elon                      BUS
        Wayne, Bruce                    ELEC
        Buffet, Warren                  BUS
        Turing, Alan                    MATH
        Bueller, Ferris                 BUS
Total Enrollment: 16
```

```
Students enrolled in Signal Processing and Communications taught by Electronic Man
        NAME                            DEPT
        --------------------------------------------
        Potter, Harry                   COMP
        Grande, Ariana                  COMP
        Greenleaf, Legolas              MATH
        Clarke, Edith                   ELEC
        Caesar, Julius                  ELEC
        Gates, Bill                     COMP
        Da Vinci, Leonardo              COMP
        Einstein, Albert                COMP
        Callas, Maria                   COMP
        Trump, Donald                   BUS
        Wayne, Bruce                    ELEC
        Buffet, Warren                  BUS
        Turing, Alan                    MATH
        Tesla, Nikola                   ELEC
        Bueller, Ferris                 BUS
        Lazy, Lazzy                     COMP
Total Enrollment: 16

Students enrolled in Probability taught by Isaac Newton
        NAME                            DEPT
        --------------------------------------------
        Potter, Harry                   COMP
        Grande, Ariana                  COMP
        Greenleaf, Legolas              MATH
        Clarke, Edith                   ELEC
        Caesar, Julius                  ELEC
        Jobs, Steve                     COMP
        Gates, Bill                     COMP
        Da Vinci, Leonardo              COMP
        Einstein, Albert                COMP
        Callas, Maria                   COMP
        Redford, Robert                 MATH
        Trump, Donald                   BUS
        Wayne, Bruce                    ELEC
        Buffet, Warren                  BUS
        Newton, Isaac                   MATH
        Turing, Alan                    MATH
        Tesla, Nikola                   ELEC
        Bueller, Ferris                 BUS
Total Enrollment: 18
```
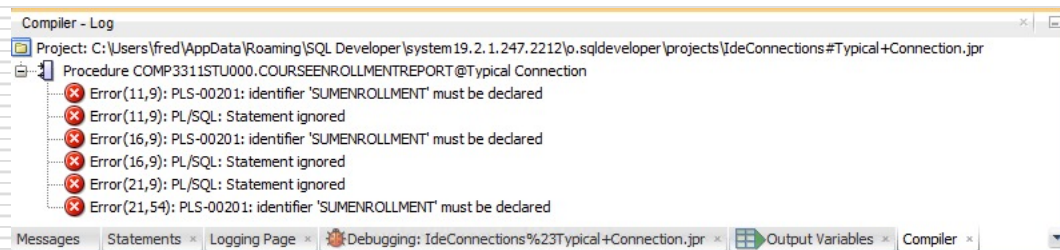
Code Editor Toolbar

☐ The **PL/SQL** Code Editor toolbar contains the following buttons (among others).

Compile compiles the procedure/function possibly for debugging. A procedure/function is also automatically compiled whenever it is saved. Any compile errors are displayed in the log of the Compiler tab shown below.

Run invokes the Run PL/SQL dialog allowing selection of the procedure/function to run and displaying a list of parameters for the selected procedure/function.
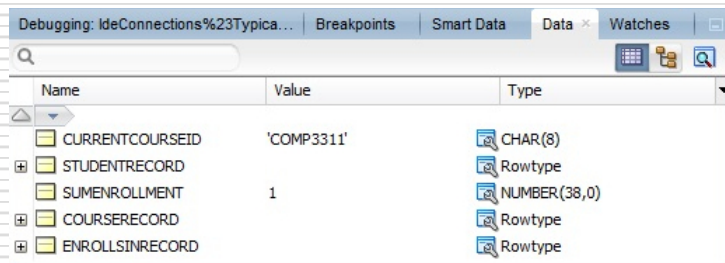
Debug runs the procedure/function in debug mode.

Running A Procedure/Function

☐ To run a procedure/ function, click the ▷ (**Run**) button or the 🐞 (**Debug**) button. Note that a procedure/function must be compiled as Compile for Debug to be run in debug mode.

☐ The Run PL/SQL dialog appears as shown on the right where the values of any required parameters can be provided.

☐ Click the **OK** button to run the procedure.
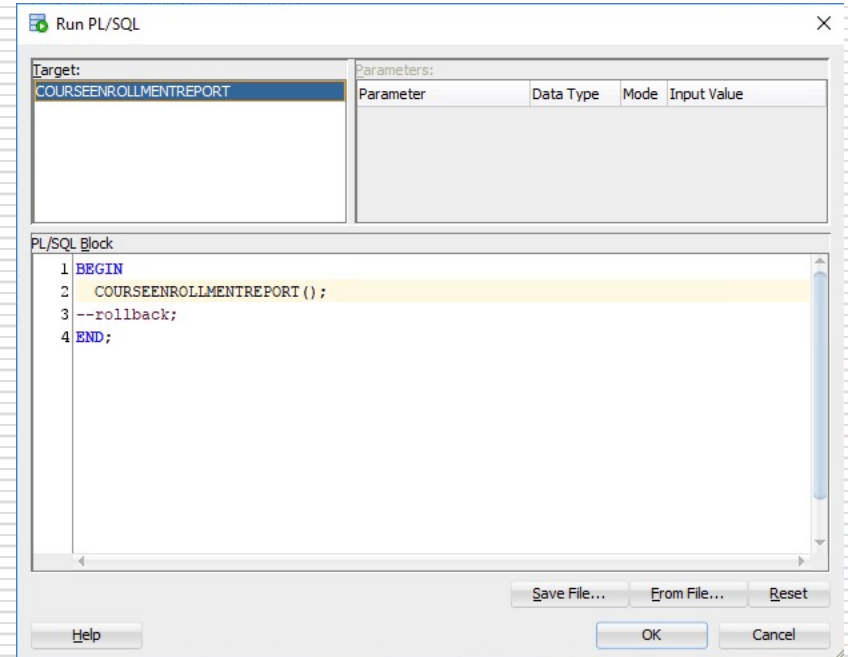
Enabling Debugging

☐ To enable debugging:

- ■ open the Preferences dialog.

- ■ select the Debugger tab.

- ■ select the checkbox Debugging Port Range.

- ■ click OK.
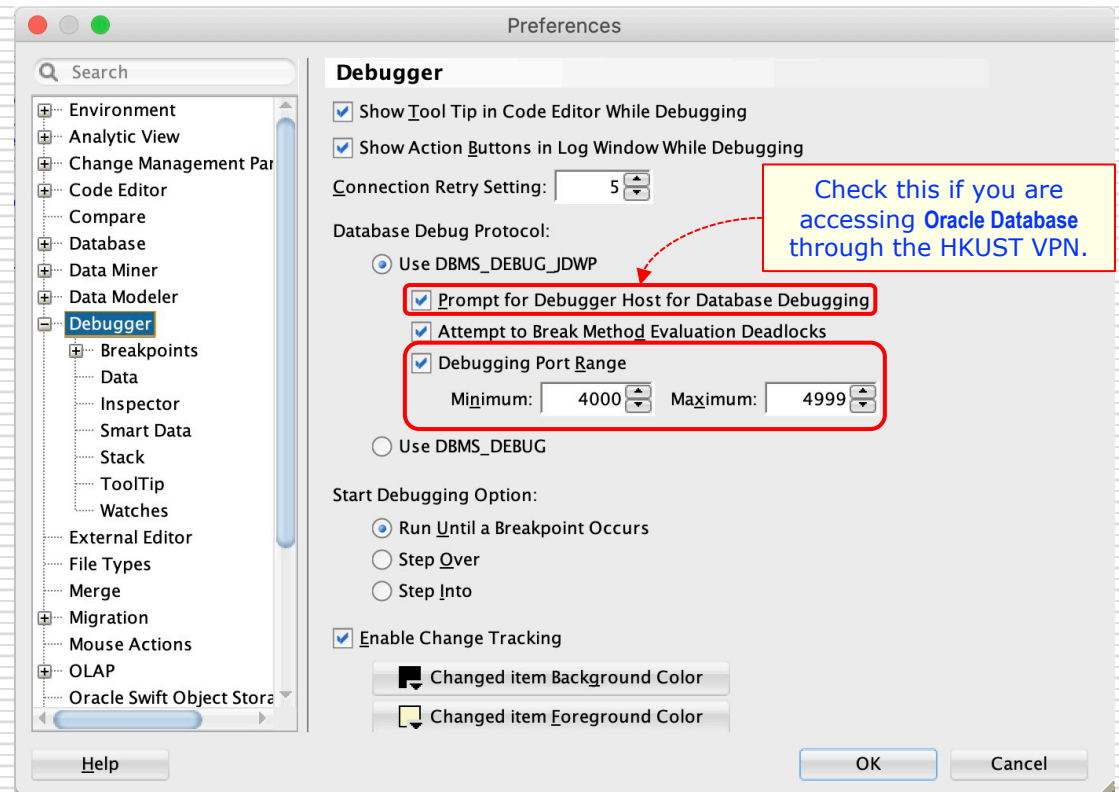
IMPORTANT
It is also necessary to configure your computer's firewall to allow incoming connections from **SQL Developer**.

Debugging With Breakpoints (1)

□ To set a breakpoint for debugging, in the line gutter, select the line number of the statement where you want to pause execution as shown in the figure.

□ Click the 🐞 (**Debug**) button to run the procedure/ function in debug mode.

Debugging With Breakpoints (2)

☐ When execution pauses at a breakpoint, the values of variables, including cursors, can be inspected either by hovering the mouse pointer over the variable (figure (a)) or by inspecting the Data tab (figure (b)) or Smart Data tab[1].

(a) inspecting variable values by hovering the mouse cursor over the variable.

(b) inspecting variable values in the Data tab.

1. The Smart Data tab shows only the values of the variables, while the Data tab shows all the data manipulated in the procedure.

Debugging With Breakpoints (3)
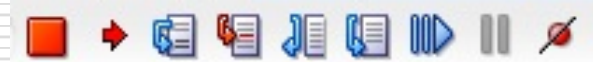
□ In the Debugging toolbar, the following buttons are available.

- **Terminate** stops the debugging session.

- **Find Execution Point** moves the cursor to where the execution has stopped.

- **Step Over** moves to the next line in the code.

- **Step Into** steps into the line of code selected, causing the debugger to continue inside the method or function of the current line of code.

- **Step Out** steps out of the current method or function and returns to the level above.
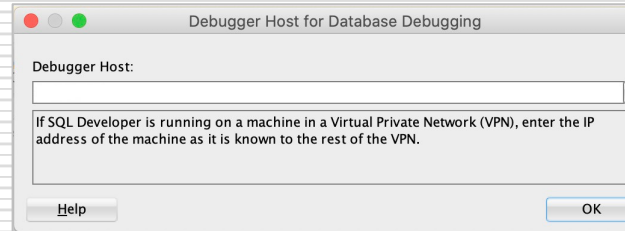
- **Step To End Of Method** goes to the end of the method.

- **Resume** continues execution until another error or breakpoint is reached.

- **Pause** pauses the debugger at its current statement.

- **Suspend All Breakpoints** turns off all breakpoints in the current procedure/ function.

Debugging With Breakpoints (4)

- ☐ To allow debugging when connecting to **Oracle Database** through the HKUST VPN, you need to check Prompt for Debugger Host for Database Debugging in the Debugger tab of the Preferences dialog as shown on slide 26.

- ☐ Then, when selecting **Debug**, you will be prompted to input your HKUST VPN IP address as shown in the dialog on the right.

- ☐ To obtain your HKUST VPN IP address, open **Pulse Secure**.
  - ◼ In the **Pulse Secure** menu select File→Connections→Advanced Connection Details…..
  - ◼ Copy the Assigned IPV4 address.
  - ◼ Paste the Assigned IPV4 address into the Debugger Host field of the Debugger Host for Database Debugging dialog shown above.