COSC344
Database Theory and Applications

Lecture 10
PL/SQL

Learning Objectives of This Lecture

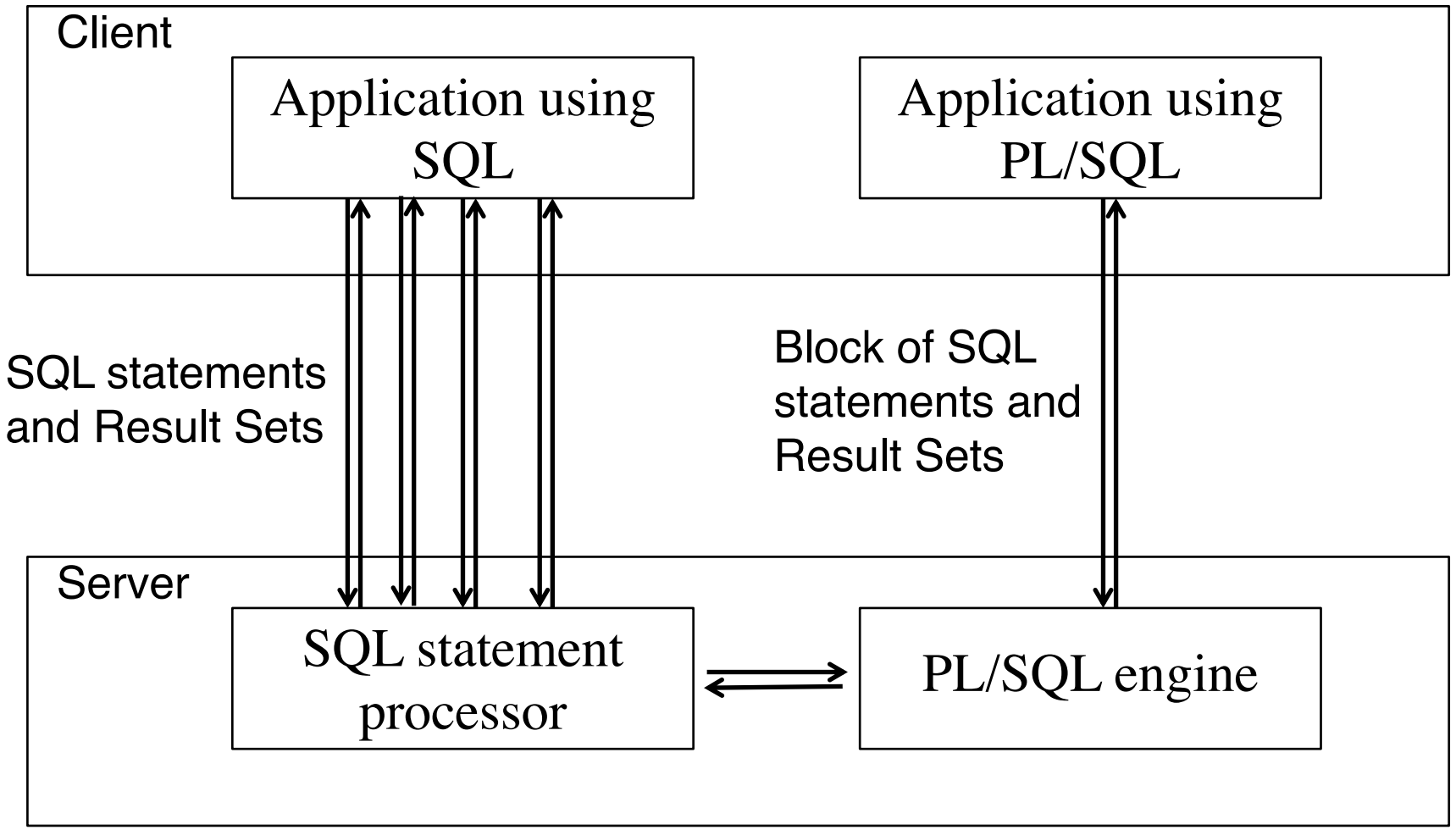
- You should
  - understand the features of PL/SQL in comparison with SQL
  - be able to distinguish between anonymous blocks and stored subprograms
  - be able to write PL/SQL functions and procedures
  - understand the handling of exceptions in PL/SQL

- Source
  - Source: Lecture notes,
    Oracle 16 documentation

What Is PL/SQL (1)

- PL/SQL (Procedural Language/Structured Query Language)
  - Oracle's procedural extension of SQL
  - An advanced fourth-generation programming language
- Features
  - Add programming capabilities to SQL
    - Control structures
    - Data abstraction
    - Information hiding
    - Exception handling
  - Enforce complex constraints
    - Set up business rules or procedures
  - Subprograms can be stored in the database
    - Compiled once and stored in executable form
    - Better performance

Comparison Between SQL and PL/SQL



What Is PL/SQL (2)

- PL/SQL is a *block-structured* language
  - Each logical block generally corresponds to a problem or a subproblem
- Basic units
  - Anonymous blocks
    - PL/SQL program units that have no name
    - Not stored in the database
    - Compiled each time they are executed
  - Stored Subprograms
    - named PL/SQL program units
    - compiled separately and stored permanently in an Oracle database,
      - » Procedures: with no returned value
      - » Functions: with a returned value

Anonymous Blocks

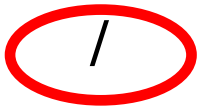
- The Structure of an Anonymous Block

```
[ DECLARE
   ... optional declaration statements ... ]

BEGIN
   ... executable statements ...

[ EXCEPTION
   ... optional exception handler statements ... ]

END;
/
```



→ Tell the PL/SQL engine to execute the block

Variable Declarations

- Syntax: <variable> <datatype>;
  - SQL datatype such as CHAR, DATE, NUMBER
  - PL/SQL datatype such as BOOLEAN, BINARY_INTEGER
  - e.g. part_no NUMBER(4);
- Variables can have attributes
  - A percent sign (%) serves as the attribute indicator.
  - %TYPE
    - Provides the datatype from the table
    - Syntax: <variable> <table>.<column>%TYPE;
    - e.g. fn employee.fname%TYPE;
  - %ROWTYPE
    - Provides a record type that represents a row in a table
    - Syntax: <record variable> <table>%ROWTYPE;
    - e.g. emp employee%ROWTYPE;
    - To get the components of the row: `emp.ird`

Control Structure – Conditional Control

- IF - THEN

```
IF <condition> THEN
   <command>
END IF
```

- IF - THEN - ELSE

```
IF <condition> THEN
   <command 1>
ELSE
   <command 2>
END IF
```

- IF - THEN -ELSIF

```
IF <condition 1> THEN
   <command 1>
ELSIF <condition 2> THEN
   <command 2>
ELSE
   <command 3>
END IF;
```

Control Structure – Iterative Control (1)

- Simple loop
  - Two ways to exit the loop

```
LOOP
  <statements>
  IF condition THEN
    EXIT;
  END IF;
END LOOP;
```

```
LOOP
  <statements>
  EXIT WHEN <condition>;
END LOOP;
```

- While-Loop

```
WHILE <condition> LOOP
  <statements>
END LOOP;
```

- For-Loop

```
FOR counter IN lower..upper LOOP
  <statements>
END LOOP;
```

Cursor FOR Loops

- A cursor FOR loop is a loop that is associated with (actually defined by) an explicit cursor.

```
FOR record_index in cursor_name
LOOP
  <statements>
END LOOP;
```

```
DECLARE
  CURSOR ec IS
    SELECT * FROM employee;
  emp ec%ROWTYPE;
BEGIN
  FOR emp IN ec
  LOOP
    DBMS_OUTPUT.PUT_LINE(emp.fname);
  END LOOP;
END;
/
```

Terminal Response

- Enable to display the output from PL/SQL programs
  - SET SERVEROUTPUT ON;
- Display the outputs
  - DBMS_OUTPUT.PUT(); /* put a partial line in the buffer */

```
DBMS_OUTPUT.PUT( fname);
DBMS_OUTPUT.PUT( '    ' );
DBMS_OUTPUT.PUT( lname);
DBMS_OUTPUT.NEW_LINE;
```

  - DBMS_OUTPUT.PUT_LINE(); /* put a complete line in buffer*/

```
DBMS_OUTPUT.PUT_LINE (fname ll '       ' ll lname);
DBMS_OUTPUT.PUT_LINE ('Salary is: ' ll salary);
```

Exception Handling

- System defined exceptions
  - NO_DATA_FOUND
  - ZERO_DIVIDE
  - INVALID_CURSOR
  - OTHERS
  - (refer to the PL/SQL user's guide)
- User defined exceptions
  - Must be declared in the declarative part of a PL/SQL block

```
DECLARE
   toomuch EXCEPTION;
```

  - Must be raised explicitly by RAISE statement.

```
IF emp.salary>50000 THEN
      RAISE toomuch;
END IF;
```

Anonymous Block Example

Retrieve and display the first and last names of all employees. If the salary of an employee is larger than 50000, raise an exception.

```
DECLARE
  CURSOR ec IS
    SELECT * FROM employee;
  emp ec%ROWTYPE;
  toomuch EXCEPTION;
BEGIN
  FOR emp IN ec LOOP
    DBMS_OUTPUT.PUT_LINE(emp.fname || ' ' || emp.lname);
    IF emp.salary>50000 THEN
      RAISE toomuch;
    END IF;
  END LOOP;
EXCEPTION
  WHEN NO_DATA_FOUND THEN NULL;
  WHEN toomuch THEN
    DBMS_OUTPUT.PUT_LINE('Someone makes too much');
END;
/
```

Save the script in a file named `getname.sql`

SQL> `@getname.sql`

NULL statement: it does nothing other than pass control to the next statement.

What Is PL/SQL (2)

- PL/SQL is a *block-structured* language
  - Each logical block generally corresponds to a problem or a subproblem
- Basic units
  - Anonymous blocks
    - PL/SQL program units that have no name
    - Not stored in the database
    - Compiled each time they are executed
  - Stored Subprograms
    - named PL/SQL program units
    - compiled separately and stored permanently in an Oracle database,
      - » Procedures: with no returned value
      - » Functions: with a returned value

Procedure (1)

- Basic Syntax

```
CREATE [OR REPLACE] PROCEDURE name
  [(argument [IN|OUT|IN OUT] datatype
  [{,argument [IN|OUT|IN OUT] datatype
      }] )]
IS|AS
  /* declaration section */
BEGIN

  /* executable section – required */

[EXCEPTION

  /* error handling statements */]

END [name];
/
```

Procedure (2)

- Specifying parameter modes
  - IN (default)
    - Pass values to the procedure being called
    - The value of an IN parameter can not be changed inside the procedure (acting like a constant)
  - OUT
    - Return values to the caller of the procedure
    - Inside the procedure, the value of an OUT parameter can be changed or referenced in any way (acting like a variable)
  - IN OUT
    - Pass initial values to the procedure being called and return updated values to the caller
    - The value of an IN OUT parameter can be changed inside the procedure (acting like an initialized variable)

Procedure (3)

- Creating a procedure (from a file)
  - @<filename>.sql e.g., @empByIRD.sql
  - The program will be compiled and the procedure will be created and stored in the database
- Executing a procedure
  - EXECUTE <name>(<parameters>)
    - EXECUTE empByIRD('123456789');
  - EXECUTE <name>;
    - EXECUTE toomuch;

Procedure Example

```
CREATE OR REPLACE
PROCEDURE insertY (aa IN NUMBER)
AS
  bb NUMBER := 5;
BEGIN
  IF aa > bb THEN
    INSERT INTO y VALUES (aa, 1);
  ELSE
    INSERT INTO y VALUES (bb, 2);
  END IF;
END;
/
```

Assuming there is a table y

JN1	JN2
3	7
9	8
6	1
5	2

```
SQL>  EXEC insertY (6);

SQL>  EXEC insertY (4);
```

Functions

- Functions and procedures are structured alike, except that functions have a RETURN clause.
- Basic syntax

```
CREATE [OR REPLACE] FUNCTION name
  [(argument datatype
  [{,argument datatype}] )]
RETURN datatype
IS|AS
  /* declaration section */
BEGIN
  /* executable section – required */
EXCEPTION
  /* error handling statements */
END[name];
/
```

No parameter mode
IN OUT

Function Example

```
CREATE OR REPLACE
FUNCTION getBDate (v_ird VARCHAR2)
RETURN DATE
AS
  v_bdate employee.bdate%TYPE;

BEGIN
  SELECT bdate
    INTO v_bdate
    FROM employee
    WHERE ird = v_ird;
  RETURN v_bdate;
END;
/
```

Must have a RETURN statement

Declaring Subprograms

- PL/SQL requires that all procedures and functions must be defined or declared before called.

```
DECLARE
  FUNCTION getBDate (v_ird VARCHAR2) RETURN DATE;
  bdate DATE;

BEGIN
 …
 bdate:= getBDate('123456789');
 …
END;
/
```

Compilation Errors

- Loading a procedure or function may cause compilation errors.

- PL/SQL does not always tell you about compilation errors. To show the most recent compilation error, use the following command:

  SHOW ERRORS;

- To get rid of procedures or functions:
  - DROP PROCEDURE <name>;
  - DROP FUNCTION <name>;

Packages

- Collects a group of related procedures and functions.
  - Provides for an interface.
  - Allows hidden or private functions and procedures.
- Package basic syntax

```
CREATE OR REPLACE PACKAGE package_name IS
  -- Package specification
  --  (function and procedure
  --   prototypes)
END[package_name];
/
CREATE OR REPLACE PACKAGE BODY package_name
IS
  /* Package body (function and
       procedure definitions)
  */
END[package_name];
/
```

Package Example

- /coursework/344/pickup/dep_package.sql

```
CREATE OR REPLACE PACKAGE dep_package IS
   -- Inserts a specific dependent who will be born tomorrow
   PROCEDURE insertDep;
   -- Deletes a named dependent
   PROCEDURE deleteDep (name dependent.dependent_name%TYPE);
   -- Calculates tomorrow's date
   FUNCTION nday RETURN DATE;
END dep_package;
/
CREATE OR REPLACE PACKAGE BODY dep_package IS
  PROCEDURE insertDep IS
  BEGIN
      INSERT INTO dependent VALUES
          ('453453453', 'Nicole', 'F', nday(), 'Daughter');
  END insertDep;
  PROCEDURE deleteDep (name dependent.dependent_name%TYPE) IS
  BEGIN
       DELETE FROM dependent WHERE dependent_name=name;
  END deleteDep;
  FUNCTION nday RETURN DATE IS
  BEGIN
    RETURN (SYSDATE + 1);
   END nday;
END dep_package;
/
```

To execute a function that is defined in a package, prefix the function name with the package name.

```
dep_package.insertDep
```

Summary

- PL/SQL features
- Anonymous blocks
  - Variable declaration
  - Control structures
  - Error handling
- Procedures
- Functions
- Packages