Oracle PL/SQL

John Ortiz

Overview of PL/SQL

- Oracle's Procedural Language extension to SQL.
- Support many programming language features. If-then-else, loops, subroutines.
- Program units written in PL/SQL can be compiled and stored in Oracle DB.
- Compiled subroutines can be used in SQL.
- PL/SQL code is portable across all operating systems that support Oracle.
- PL/SQL does not support DDL.

PL/SQL Block

- A PL/SQL block contains logically related SQL and PL/SQL statements.
- Three sections in a typical PL/SQL block:

```
declare
    type, variable, function, procedure, …
begin
    SQL & PL/SQL statements
exception
     exception handling
end
/        /* program end */
```


Sample Program One

- Print a message indicating if student 1234 is a CS major.

```
declare
  student_name  Students.Name%TYPE;
  student_major  Students.Major%TYPE;
begin
   select Name, Major
   into student_name, student_major
   from Students  where SID = `1234';
```

Sample Program One (cont.)

```
  if (student_major = 'CS') then
     dbms_output.put_line('A CS student.');
  else
     dbms_output.put_line('Not a CS student.');
  end if;
end;
/  /* end each PL/SQL program with / */
```


Execute PL/SQL Programs

- Save the program in a file: sample1.sql
- Execute the program in SQL*Plus

  SQL> start sample1
- Enable output to the screen:

  SQL> set serveroutput on

  or place "set serveroutput on" at the beginning of the PL/SQL program.

Declaration

```
declare
 average_GPA number(3,2);
 no_of_depts constant number(2) := 23;
 no_of_students number(5) not null := 12000;
 employee_name varchar2(30);
 state_code char(2);
 done boolean default true;
```

☛ declare one variable at a time.

PL/SQL Data Types

- Built-in Simple Types:
  - binary_integer: $-2^{31}-1$ to $2^{31}-1$
  - natural: 0 to 2^{31}
  - positive: 1 to 2^{31}
  - long: character string up to 32,760 bytes
  - boolean: boolean type (true, false, null)
  - number(n,m), char(n), varchar2(n) , date : same as their counterparts in SQL
- %type: using an existing column type.
  - v_student_gpa Students.gpa%type

Record Type & Row Type

- Define a new record type.

```
type course_record_type is record
     (course_id   Courses.cid%type;
      title   Courses.title%type;
      credit_hours   number);
  course_record  course_record_type;
```

- %rowtype: use an existing tuple type.

```
      one_student  Students%rowtype;
```

☛ Use "." to reference record fields

```
course_record.title = 'Database I';
```


Simple Statements

- Null Statement: null;
- Assignment Statement:

```
i := i + 1;
name := 'Smith';
```

- Conditional Statement:

```
if condition1 then statement1;
 elsif condition2 then statement2;
 else statement3;
end if;
```

☛ Both elsif and else are optional.

Conditions Involving Null

- Testing for null or not null as in SQL

  if (course_title is null) then …
- Expressions involving null will result a null.

  (s1.gpa > s2.gpa) results a null if s1.gpa is null
- If a condition results false or null, the corresponding statement will not be evaluated.
- Truth values involve null:
  - A and B: null if either A or B is null
  - A or B: null only if both A and B are null
  - Not A: null if A is null

Loop Statement

- Simple loop:

```
loop
  statements;
exit when condition;
end loop;
```

- While loop:

```
while condition loop
  statements;
end loop;
```

For Loop

- Syntax: for variable in [reverse] low..high loop
  statements;
  end loop;
  for x in -10..10 loop … end loop;
  for x in reverse -10..10 loop … end loop;
- No need to declare loop variable explicitly.
- Loop variable is not accessible outside loop.
- Modifying loop variable will cause an error.

Sample Program Two

- As long as the total company payroll is less than $5 million, increase employee's salary by 2%.

```
declare
   company_payroll number;
begin
   select sum(salary)
   into company_payroll
   from Employees;
```

Sample Program Two (cont.)

```
   while company_payroll < 5000000 loop
      update Employees
      set salary = salary * 1.02;
      select sum(salary)
      into company_payroll
      from Employees;
   end loop;
end;
/
```


Exceptions

- An exception is any error that occurs during program execution.
- exception /* exception section */

```
   when dup_val_on_index then
      dbms_output.put_line(sqlcode ||
         '--' || sqlerrm);
  end;
```

- Output message if the exception occurs:

  -1--ORA-00001: unique constraint violated

Exceptions

- Syntax: exception:

```
when exception_name then
   error-handling-code;
...
when others then
   error-handling-code;
```

- Pre-defined exceptions: invalid_cursor, too_many_rows, dup_val_on_index, no_data_found, etc.

User-Defined Exceptions

```
declare /* declare section */
   /* Must be explicitly declared */
  invalid_gpa exception;
begin /* execution section */
  if (gpa < 0 or gpa > 4.0) then
     /* Must be raised by user */
     raise invalid_gpa;
  end if;
```

User-Defined Exceptions (cont.)

```
exception  /* exception section */
  when invalid_gpa then
    dbms_output.put_line('GPA value is invalid.');
end;
/
```

- It is a good practice to handle all exceptions explicitly in the exception section.
  - Can improve reliability and readability

Procedure

- Assume a Customer relation. Write a PL/SQL program to retrieve a given customer. Report an exception if not found.

```
set serveroutput on
declare
  v_cid customers.cid%type;
  v_cname customers.cname%type;
  v_city customers.city%type;
  status boolean;
```

Procedure (cont.)

```
procedure get_customer(
   cust_id  in  customers.cid%type,
   cust_name  out  customers.cname%type,
   cust_city  out  customers.city%type,
   status  out  boolean) is
 begin
    select cname, city into cust_name, cust_city
    from customers where cid = cust_id;
    status := true;
 exception /* optional */
    when no_data_found then status := false;
 end; /* procedure */
```


Procedure (cont.)

```
begin /* main block */
   v_cid := 'c001';
   get_customer(v_cid, v_cname, v_city, status);
    if (status) then
      dbms_output.put_line(v_cid || ' ' ||
           v_cname || ' ' || v_city);
    else
      dbms_output.put_line('Customer ' ||
           v_cid || ' not found');
    end if;
end; /* main block */
/
```

Function

- Use a function to find the number of customers in a given city.

```
set serveroutput on
declare
   v_city customers.city%type;
   customer_no number;
```


Function (cont.)

```
function no_of_customers(
    cust_city  in  customers.city%type)
    return number is  /* must return a data */
    num_of_customers number;
begin
    select count(*) into num_of_customers
    from customers where city = cust_city;
    return (num_of_customers);
end; /* function */
```

Function (cont.)

```
begin /* main block */
  v_city := 'Dallas';
  customer_no := no_of_customers(v_city);
  dbms_output.put_line('Number of customers
       in ' || cust_city || ' is ' || customer_no);
end; /* main block */
/
```


Stored Procedures and Functions

- procedure and function can be compiled and stored for later use (in SQL and other PL/SQL blocks).
- Stored procedures and functions are created by

  create or replace procedure proc_name …

  create or replace function func_name …
- Only input parameters are allowed for functions.

A Sample Stored Procedure

```
create or replace procedure get_cus_name(
    v_cust_id in customers.cid%type) is
    v_cust_name customers.cname%type;
begin
    select cname into v_cust_name
    from customers where cid = v_cust_id;
    dbms_output.put_line('Customer
        name: ' ||  v_cust_name);
end;
/
show errors
```


Compilation & Execution

- Compile stored procedure in file proc.sql

  SQL> start proc
- show errors displays errors detected during compiling the procedure.
- Execute stored procedure get_cus_name

  SQL> execute get_cus_name('c001');