Advanced Data Base
(8trd157)
Lab4 :
(Part II: project phase 4)

PART I	Test of PL/SQL blocks, procedures and triggers with Oracle 18c
PART II	Creation of a C program using Pro*C and accessing an Oracle database

Paul Girard, Ph.D.

PART I
(personal exercise)

Objective of PART I
This part explains how to create and test PL/SQL anonymous blocks, stocked procedures and *triggers*. Some previous tables will be used and some will be created. **sqlplus** will be used in this part.

Methodology
1. Download all files to your PC using my **web site** in the following section
   ==> "**Oracle: Example of PL/SQL blocks** " .

   With EditPlus, upload the following files :.dat, .ctl, .sql and load_tables to the server dim-ensxcn1.uqac.ca and make sure that all tables of lab2 are loaded with data.

   Use a session with Putty (ssh) to test these PL/SQL blocks shown in the following sections

List of copied and uploaded files

cre_emp.sql	cretab2.sql	plsql1.sql	plsql2.sql	plsql3.sql	cre_trigger.sql
proc1.sql	proc_del_emp.sql		proc_del_part2.sql		

2. In your user schema, create the table *emp* with the file *cre_emp.sql*

cre_emp.sql

```
drop table emp;
create table emp
        (emp_num     number      primary key,
        emp_name     char(15)    not null,
        addr         char(15)    default 'Beijing');
```

execution of *cre_emp*

```
SQL> @cre_emp
        drop table emp;
        Table dropped

        Table created.
```

3. The PL/SQL block defined in the file *plsql1.sql* will generate 5 rows in the table *emp* when it will be executed. The "/" at the end of the file will start the execution automaticcaly. Read this file, try to understand its objective, then use ***sqlplus*** to execute it.

plsql1.sql

```
declare
   num  number := 100;          -- initializate num
begin
   delete from emp;    -- initialize table emp
   for i in 1..5
   loop
                insert into emp (emp_num, emp_name) values (num, concat ('employee_name',i));
                num := num + 100;
   end loop;
   commit;
end;
/
```

execution of *plsql1.sql*

```
SQL> @plsql1

PL/SQL procedure successfully completed.

SQL> select * from emp;

EMP_NUM     EMP_NAME              ADDR
----------  --------------------  ---------------
100         employee_name1        Beijing
200         employee_name2        Beijing
300         employee_name3        Beijing
400         employee_name4        Beijing
500         employee_name5        Beijing
```

4. The PL/SQL block defined in the file *plsql2.sql* will give the result of the transaction **low_part** used by the purchasing agents in a special table called *part2* which will describe all parts to be purchased. This table is created with the file *cretab2.sql* (a status of 'A' means ==> activate a purchase order). A cursor will be used because each row will have to processed.

cretab2.sql

```
drop table part2;
create table part2
        (part_id        number(5),
        part_name       char(15),
        status          char(1) default 'A');
```

plsql2.sql

```
declare
    cursor c1 is
      select part_id, part_name, stock_qty, order_qty, min_qty
  from part
  where min_qty  >= (stock_qty + order_qty)
      order by part_id asc;    -- ascending order on part_id

    my_part_id                part.part_id%TYPE;
    my_part_name              part.part_name%TYPE;
    my_stock_qty              part.stock_qty%TYPE;
    my_order_qty              part.order_qty%TYPE;
    my_min_qty                part.min_qty%TYPE;
    status                    char(1) := 'A';

begin
   delete from part2;                    -- initialize the table part2

   -- open the cursor c1, execute the select and set the pointer on the first row of the result
   open c1;

   loop
      fetch c1 into my_part_id, my_part_name, my_stock_qty, my_order_qty, my_min_qty;
      exit when c1%notfound;          /* exit from loop if no more row
                                      to process in the result of select */
      insert into part2 values (my_part_id, my_part_name, status);
      commit;
   end loop;

   close c1;                    -- close cursor
end;
/
```

execution of *plsql2.sql*

```
SQL> @cretab2
SQL> @plsql2
PL/SQL procedure successfully completed.

SQL> select * from part2;

part_id        part_name        status
----------     ---------------   -
1001           'motor 1'        A
1003           'batteries AA'   A
```

note: parts 1001 and 1003 must be purchased from a supplier.

5. The file *proc1.sql* shows an example of a local procedure called *mod_status* in the PL/SQL block:

```
declare
    my_part_id      part.part_id%TYPE;
    status          char(1);

/* Declare the local procedure mod_status*/
    procedure mod_status (num number) is
            begin
                    update part2 set status='I' where part_id = num;
            end;

begin
   my_part_id := 1001;                  -- initialize part_id of part2
   mod_status (my_part_id);
   commit;
end;
/
```

The transaction **cre_po** *(creation of a purchase order)* by an agent for the part 1001 should call the procedure *mod_status* to modify the status of this part to 'I' in the table *part2*.

execution

```
SQL> @proc1
PL/SQL procedure successfully completed.

SQL> select * from part2;

part_id      part_name       status
----------   ---------------  -
1001         'motor 1'        I
1003         'batteries AA'   A
```

6. The following example creates the stocked procedure *del_emp(num)* in the Oracle data dictionary shown by the file *proc_del_emp.sql;* this procedure is called by the PL/SQL block *plsql3.sql* without declaring it like the previous example.

proc_del_emp.sql

```
create procedure del_emp(num number) as
begin
        delete from emp where emp_num = num;
end;
/
```

plsql3.sql

```
declare
   num  number := 100;
begin
   del_emp(num);          /* delete employee 100 */
   commit;
end;
/
```

execution

```
SQL> @proc_del_emp
Procedure dropped.
Procedure created.

SQL> select * from emp;
        MAT     NOMEMP
        ---------- ---------------
        100     nom_emp1
        200     nom_emp2
        300     nom_emp3
        400     nom_emp4
        500     nom_emp5

SQL> @plsql3
PL/SQL procedure successfully completed.
SQL> select * from emp;
        MAT     NOMEMP
        ---------- ---------------
        200     nom_emp2
        300     nom_emp3
        400     nom_emp4
        500     nom_emp5
```

7. The last example shows how to create and use a **trigger** ; *tr1* will "trigger" an insert operation in the table *log_part* immediately after a modification *(insert, delete, update)* has been done on the table *part* by any user. The user code, the type of modification and the time of this modification will be recorded in the table *log_part*.

cre_trigger.sql

```
drop table log_part;
drop trigger tr1;
create table log_part
        (user_code      char(12),
        type            char(8),
        hour            char(20));

create trigger tr1 after update or delete or insert on part for each row
declare
        typ     char(8);
        hr      char(20);
begin
        if updating then typ := 'update';       end if;
        if deleting then typ := 'delete';       end if;
        if inserting then typ := 'insert';      end if;
        hr := to_char(sysdate,'dd-mm-yyyy hh:mm:ss');
        insert into log_part values(user,typ, hr);
end;
/
```

Creation and activation of *tr1* after an *update* to part

```
SQL> @cre_trigger
      table created
      trigger created

SQL> update part set min_qty=12 where part_id=1004;
      1 row updated.

SQL> select * from log_part;

USER_CODE        TYPE        HOUR
------------     --------    -------------------------
PGIRARD          update      06-04-2001 09:04:19
```

===>User PGIRARD has made an *update* to the table *part*.

<u>If you created a trigger, execute a drop trigger ** immediately after your test because these programs overload heavily the server.</u>

Now you are ready to continue your project with Part II

PART II *(project phase 4)*

This part should be tested

1. For phase IV, each user type will create his own program in C on dim-ensxcn1.uqac.ca; this program will execute some transactions already tested in lab3.

2. First download the basic C program and the compilation procedures from the web site. The annex 1 shows the source listing of this program and its execution.

3. List of transactions you have to do

User type technician:	**lispart**, **explosion**
User type PA:	**low_part, pot_supp**
User type SK:	**part_out, list_po**
User type PDS:	**modresp, quant**
User type SKS:	**invent, value**

If your team has 3 students, one should do the 2 PA transactions, one the 2 SK transactions and the last one supporting in one C program the following 4 transactions : explosion, implosion, modresp, quant

Annex 1

Listing of the program *lab4base.pc* accessing Oracle 8i, 9i, 11g, 18c

```
/*
*       Author:         Paul Girard Ph.D., UQAC
*       Date:           May 2019
*       Objective:      Program using Pro*C/C++ and gcc to show how to use
*                       different type of PL/SQL blocks & SQL in a C PGM
*
*       Step 1: Precompilation with Pro*C/C++
* proc INAME=lab4base.pc CODE=ANSI_C SQLCHECK=semantics MODE=oracle USERID=user/password CHAR_MAP=charz
*
*       Step 2: Compilation with gcc
* gcc lab4base.c -o lab4base -include /$ORACLE_HOME/precomp/public/sqlca.h -lclntsh -L$ORACLE_HOME/lib/ -B$ORACLE_HOME/lib
*/

#include <stdio.h>
#include <stdlib.h>

void sql_error();
void do_connect();
void lispart();
void respon();
void modstat();
void deletion();
int print_menu();

int main()
{
        EXEC SQL WHENEVER SQLERROR do sql_error("Error at connect");

        do_connect();           /* connection to Oracle instance */

/*      Display the program menu
*       and execution of the transaction
*/

        while (1)       /* infinite loop */
        {
         switch (print_menu())
         {
          case 1: lispart();
                  break;
          case 2: respon();
                  break;
          case 3: modstat();
                  break;
          case 4: deletion();
                  break;
          case 5: print_menu();
                  break;
         case 6: puts("\nAu revoir Sayonara Bye bye, Ni Hao");
                 exit(0);
          default: puts("\n =====> Enter a digit from the menu please ?");
```

```
                    break;
          }
          }

          EXEC SQL COMMIT WORK RELEASE;          /* libère les verroux et déconnecte */

          exit(0);
}                         /* end of main() program */

/* ***********************************************************************************
*         Function to display the original error Oracle message
*  ***********************************************************************************
*/

void sql_error(char *msg)
{
          char ora_msg[512];                     /* buffer for Oracle error message */
          int buf_len, msg_len;

          EXEC SQL WHENEVER SQLERROR continue;   /* Prevent an infinite loop */

          printf("\n%s\n",msg);                  /* print the local program message */
          buf_len=sizeof(ora_msg);

          sqlglm(ora_msg,&buf_len, &msg_len);    /* read the Oracle error message */
          printf("\n%.*s\n",msg_len, ora_msg);   /* print this message */

          EXEC SQL ROLLBACK RELEASE;             /* free locks */
          exit(1);
}                /*        fin de sql_error */

/* ***********************************************************************************
*         Function to do a connection to an Oracle user schema
*  ***********************************************************************************
*/

void do_connect()
{
  char *uid="userid/password";

          EXEC SQL CONNECT :uid;
          printf("Connected to Oracle schema\n");
}

/* ***********************************************************************************
*         Function to display the contents of a given part
*  ***********************************************************************************
*/

void lispart()
{
char      description[16];    /* C needs 1 more octet than Oracle for binary 0
                                 at the end of a C character type */
int       my_partid, none=1;
          for (;;)            /* infinite loop until user enter a 0 */
          {
```

```
                printf("Number of the part (0 to quit )? ");
                scanf("%d",&my_partid);

                printf("Part Number : %d\n",my_partid);
                if (my_partid == 0)
                {
                        EXEC SQL COMMIT;
                        printf("End of this transaction\n");
                        break;
                }

/*-------------- Beginning the PL/SQL block -----------------*/

        EXEC SQL EXECUTE
        BEGIN
                SELECT part_name INTO :description      /* note: description needs 1 octet more */
                FROM part WHERE part_id = :my_partid;
                EXCEPTION
                WHEN NO_DATA_FOUND THEN
                :none:=0;
        END;
        END-EXEC;

/*-------------- end of PL/SQL block -----------------*/

        if (none==0)
          {
        printf("Record not found \n");
                none=1;
          }
        else
           {
                printf("Name of the part\n");
                printf("---------------\n");
                printf("%s\n",description);
           }
        EXEC SQL COMMIT;
        }                                       /* end of infinite loop */
        return;
}

/* ***************************************************************
*       Function to display each part with the responsible agent
*  ***************************************************************
*/

void respon()
{
int     my_emp_num, my_partid;
char    description[16];   /* 1 more octet for the binary 0 */
char    my_pa_name[16];            /* 1 more octet for the binary 0 */

        EXEC SQL declare c1 cursor for
        select p.part_id, p.part_name, pa.emp_num, pa.pa_name
                from part p, responsible r, pa_agent pa
                where pa.emp_num=r.emp_number and p.part_id=r.part_number;
```

```
        EXEC SQL open c1;

        EXEC SQL WHENEVER NOT FOUND do break;

        printf("\tPart ID\tName of Part\t\tEmp Number\tAgent\n");
        for (;;)
        {
                EXEC SQL fetch c1 into :my_partid, :description, :my_emp_num, :my_pa_name ;

                printf("\t%4d\t%s\t\t%4d\t%s\n",my_partid, description,my_emp_num,my_pa_name);
        }

        printf("End of this transaction\n");
        EXEC SQL close c1;
        EXEC SQL COMMIT;                /* free locks and keep the connection */
        return;
}

/* ***********************************************************************************************
*       Function modifying the status of a part
*  ***********************************************************************************************
*/

void modstat()
{
int     num, status;
        for (;;)
        {
                printf("Enter the part id to change its status (0 to exit)? ");
                scanf("%d",&num);
                if (num == 0)
                {
                        printf("End of this transaction\n");
                        return;
                }

/*-------------- beginning of PL/SQL block with local procedure  -----------------*/

                EXEC SQL EXECUTE
                DECLARE
                procedure mod_status(no IN integer, stat OUT integer) is
                BEGIN
                        stat := 1;
                        update part2 set status='I' where part_id = no;
                        if SQL%NOTFOUND then stat:=0;
                        end if;
                END;
                BEGIN           -- execution part
                        mod_status(:num, :status);
                        commit;
                END;
                END-EXEC;

/*-------------- end of PL/SQL block -----------------*/
```

```
        if (status==0)
          {
        printf("Record not found \n");
          }
        else      printf("Modification done\n");
        }
        return;
}

/* ******************************************************************************
*       Function to delete a part from the table part2 ; the stocked procedure
*       must be created before calling this block
*  ******************************************************************************
*/

void deletion()
{
int     num, stat=1;
        for (;;)
        {
        printf("\nEnter the number of the part in part2 to be deleted (0 to quit)? ");
        scanf("%d",&num);
        if (num == 0)
        {
                printf("End of this transaction\n");
                return;
        }
        EXEC SQL EXECUTE
        BEGIN
                del_part2(:num);
                if SQL%NOTFOUND then :stat:=0;
                end if;
                commit;
        END;
        END-EXEC;
        if (stat==0)
        {
     printf("part not found \n");
        }
        else      printf("Part deleted\n");
        }
        return;
}

/* ************************************************************
*       Function print menu and choose transaction
*  ************************************************************
*/

int print_menu()
{
  int choice;                           /* id of menu */
        printf("\n\t Choose a transaction by entering a number \n");
        printf("\t ******************************************\n");
        printf("\t (1) LISPART\n");
        printf("\t (2) RESPON\n");
```

```
	printf("\t (3) MODSTAT\n");
	printf("\t (4) DELETION\n");
	printf("\t (5) DISPLAY MENU\n");
	printf("\t (6) QUIT\n");
	printf("Enter your choice ? ");
	scanf("%d",&choice);		/* read the choice */
	return choice;
}
```

Execution

```
dim-ensxcn1:pgirard> chmod 700 precompile
dim-ensxcn1:pgirard> chmod 700 compile

dim-ensxcn1:pgirard> ./precompile

Pro*C/C++: Release 8.1.7.0.0 - Production on Sat Mar 8 17:48:12 2008

(c) Copyright 2000 Oracle Corporation.  All rights reserved.

System default option values taken from: /disk/disk1/oracle/OraHome1/precomp/adg

dim-ensxcn1:pgirard> ./compile

dim-ensxcn1:pgirard> ./lab4base
Connected to Oracle schema

        Choose a transaction by entering a number
        ****************************************
        (1) LISPART
        (2) RESPON
        (3) MODSTAT
        (4) DELETION
        (5) DISPLAY MENU
        (6) QUIT

Enter your choice ? 1
Number of the part (0 pour finir)? 1001
Part Number : 1001
Name of the part
---------------
'motor 1'
Number of the part (0 pour finir)? 0
End of this transaction

        Choose a transaction by entering a number
        ****************************************
        (1) LISPART
        (2) RESPON
        (3) MODSTAT
        (4) DELETION
        (5) DISPLAY MENU
        (6) QUIT

Enter your choice ? 2
       Part ID Name of Part        Emp Number      Agent
       1001    'motor 1'             100           Tom
       1002    'motor 2'             100           Tom
       1003    'batteries AA'        101           Bruce
       1004    'batteries 90C'       101           Bruce
       1005    'spark plug 1'        102           Bobbie
       1006    'spark plug 2'        102           Bobbie
End of this transaction

        Choose a transaction by entering a number
        ****************************************
```

```
        (1) LISPART
        (2) RESPON
        (3) MODSTAT
        (4) DELETION
        (5) DISPLAY MENU
        (6) QUIT

Enter your choice ? 3
Enter the part id to change its status (0 to exit)? 1001
Modification done
Enter the part id to change its status (0 to exit)? 0
End of this transaction

        Choose a transaction by entering a number
        ****************************************
        (1) LISPART
        (2) RESPON
        (3) MODSTAT
        (4) DELETION
        (5) DISPLAY MENU
        (6) QUIT

Enter your choice ? 4

Enter the number of the part in part2 to be deleted (0 to quit)? 1001
Part deleted

Enter the number of the part in part2 to be deleted (0 to quit)? 0
End of this transaction

        Choose a transaction by entering a number
        ****************************************
        (1) LISPART
        (2) RESPON
        (3) MODSTAT
        (4) DELETION
        (5) DISPLAY MENU
        (6) QUIT

Enter your choice ? 6

Au revoir Sayonara Bye bye, Ni Hao
dim-ensxcn1:pgirard>
```