A
Practice Solutions

Oracle University and En-Sof Informatica E Treinamento Ltda use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 1: Solutions

PL/SQL Basics

1. What are the four key areas of the basic PL/SQL block? What happens in each area?
   Header section: Names the program unit and identifies it as a procedure, function, or package; also identifies any parameters that the code may use
   Declarative section: Area used to define variables, constants, cursors, and exceptions; starts with the keyword IS or AS Executable section: Main processing area of the PL/SQL program; starts with the keyword BEGIN Exception handler section: Optional error handling section; starts with the keyword EXCEPTION

2. What is a variable and where is it declared?
   Variables are used to store data during PL/SQL block execution.
   You can declare variables in the declarative part of any PL/SQL block, subprogram, or package. Declarations allocate storage space for a value, specify its data type, and name the storage location so that you can reference it. Declarations can also assign an initial value and impose the `NOT NULL` constraint on the variable.
   Syntax: `variable_name datatype[(size)][:= initial_value];`

3. What is a constant and where is it declared?
   Constants are variables that never change. Constants are declared and assigned a value in the declarative section, before the executable section.
   Syntax: `constant_name CONSTANT datatype[(size)] := initial_value;`

4. What are the different modes for parameters and what does each mode do?
   There are three parameter modes: IN, OUT, and IN OUT. IN is the default and it means a value is passed into the program. The OUT mode indicates that the subprogram is passing a value generated in the subprogram out to the calling environment. The IN OUT mode means that a value is passed into the subprogram. The subprogram may change the value and pass the value out to the calling environment.

5. How does a function differ from a procedure?
   A function must execute a RETURN statement that returns a value. Functions are called differently than procedures. They are called as an expression embedded within another command. Procedures are called as statements.

6. What are the two main components of a PL/SQL package?
   The package body and package specification

   a. In what order are they defined?
      First the package specification and then the package body

   b. Are both required?
      No, only a package specification is required. A specification can exist without a body, but a body cannot exist as valid without the specification.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED
Oracle University and En-Sof Informatica E Treinamento Ltda use only

Practice 1: Solutions (continued)

7. How does the syntax of a SELECT statement used within a PL/SQL block differ from a SELECT statement issued in SQL*Plus?
   The INTO clause is required with a SELECT statement that is in a PL/SQL subprogram.

8. What is a record?
   A record is a composite type that has internal components, which can be manipulated individually. Use the RECORD data type to treat related but dissimilar data as a logical unit.

9. What is an index-by table?
   Index-by tables are a data structure declared in a PL/SQL block. It is similar to an array and made of two components, the index and the data field. The data field is a column of a scalar or record data type, which stores the INDEX BY table elements.

10. How are loops implemented in PL/SQL?
    Looping constructs are used to repeat a statement or sequence of statements multiple times. PL/SQL has three looping constructs:

    - **Basic loops that perform repetitive actions without overall conditions**
    - **FOR loops that perform iterative control of actions based on a count**
    - **WHILE loops that perform iterative control of actions based on a condition**

11. How is branching logic implemented in PL/SQL?
    You can change the logical flow of statements within the PL/SQL block with a number of control structures. Branching logic is implemented within PL/SQL by using the conditional IF statement or CASE expressions.

Cursor Basics

12. What is an explicit cursor?
    The Oracle server uses work areas, called private SQL areas, to execute SQL statements and to store processing information. You can use PL/SQL cursors to name a private SQL area and access its stored information. Use explicit cursors to individually process each row returned by a multiple-row SELECT statement.

13. Where do you define an explicit cursor?
    A cursor is defined in the declarative section.

14. Name the five steps for using an explicit cursor.
    Declare, Open, Fetch, Test for existing rows, and Close

15. What is the syntax used to declare a cursor?
    CURSOR cursor_name IS SELECT_statement

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED
Oracle University and En-Sof Informatica E Treinamento Ltda use only

16. What does the FOR UPDATE clause do within a cursor definition?
The FOR UPDATE clause locks the rows selected in the SELECT statement definition of the cursor.

17. What command opens an explicit cursor?
```
OPEN cursor_name;
```

18. What command closes an explicit cursor?
```
CLOSE cursor_name;
```

19. Name five implicit actions that a cursor FOR loop provides.
Declares a record structure to match the select list of the cursor; opens the cursor, fetches from the cursor, exits the loop when the fetch returns no row, and closes the cursor

20. Describe what the following cursor attributes do:
%ISOPEN: Returns a Boolean value indicating whether the cursor is open
%FOUND: Returns a Boolean value indicating whether the last fetch returned a value
%NOTFOUND: Returns a Boolean value indicating whether the last fetch did not return a value
%ROWCOUNT: Returns an integer indicating the number of rows fetched so far

Exceptions

21. An exception occurs in your PL/SQL block which is enclosed in another PL/SQL block. What happens to this exception?
Control is passed to the exception handler. If the exception is handled in the inner block, processing continues to the outer block. If the exception is not handled in the inner block, an exception is raised in the outer block and control is passed to the exception handler of the outer block. If neither the inner nor the outer block traps the exception, the program ends unsuccessfully.

22. An exception handler is mandatory within a PL/SQL subprogram. (True/False)
False

23. What syntax do you use in the exception handler area of a subprogram?
```
EXCEPTION
           WHEN named_exception THEN
                statement[s];

           WHEN others THEN
                statement[s];
END;
```

Oracle University and En-Sof Informatica E Treinamento Ltda use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 1: Solutions (continued)

24. How do you code for a NO_DATA_FOUND error?

```
EXCEPTION
           WHEN no_data_found THEN
                statement[s];
END;
```

25. Name three types of exceptions.
    User-defined, Oracle server predefined, and Oracle server non-predefined

26. To associate an exception identifier with an Oracle error code, what pragma do you use and where?
    Use the `PRAGMA EXCEPTION_INIT` and place the `PRAGMA EXCEPTION_INIT` in the declarative section.

27. How do you explicitly raise an exception?
    Use the `RAISE` statement or the `raise_application_error` procedure.

28. What types of exceptions are implicitly raised?
    All Oracle server exceptions (predefined and non-predefined) are automatically raised.

29. What does the RAISE_APPLICATION_ERROR procedure do?
    Enables you to issue user-defined error messages from subprograms.

Dependencies

30. Which objects can a procedure or function directly reference?
    Table, view, sequence, procedure, function, package specification, object specification, and collection type

31. What are the two statuses that a schema object can have and where are they recorded?
    The `user_objects` dictionary view contains a column called status. Its values are `VALID` and `INVALID`.

32. The Oracle server automatically recompiles invalid procedures when they are called from the same _____. To avoid compile problems with remote database calls, we can use the _____ model instead of the timestamp model.
    database
    signature

33. What data dictionary contains information on direct dependencies?
    `user_dependencies`

34. What script do you run to create the views deptree and ideptree?
    You use the `utldtree.sql` script.

Oracle University and En-Sof Informatica E Treinamento Ltda use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

35. What does the `deptree_fill` procedure do and what are the arguments that you need to provide?
    The `deptree_fill` procedure populates the `deptree` and `ideptree` views to display a tabular representation of all dependent objects, direct and indirect. You pass the object type, object owner, and object name to the `deptree_fill` procedure.

36. What does the `dbms_output` package do?
    The `dbms_output` package enables you to send messages from stored procedures, packages, and triggers.

37. How do you write "This procedure works." from within a PL/SQL program by using `dbms_output`?
    ```
    DBMS_OUTPUT.PUT_LINE('This procedure works.');
    ```

38. What does `dbms_sql` do and how does this compare with Native Dynamic SQL?
    `dbms_sql` enables you to embed dynamic DML, DDL, and DCL statements within a PL/SQL program. Native dynamic SQL allows you to place dynamic SQL statements directly into PL/SQL blocks. Native dynamic SQL in PL/SQL is easier to use than `dbms_sql`, requires much less application code, and performs better.

Oracle University and En-Sof Informatica E Treinamento Ltda use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 2: Solutions

1. Determine the output of the following code snippet.

```
SET SERVEROUTPUT ON
BEGIN
  UPDATE orders SET order_status = order_status;
  FOR v_rec IN ( SELECT order_id FROM orders )
    LOOP
      IF SQL%ISOPEN THEN
        DBMS_OUTPUT.PUT_LINE('TRUE - ' || SQL%ROWCOUNT);
      ELSE
        DBMS_OUTPUT.PUT_LINE('FALSE - ' || SQL%ROWCOUNT);
      END IF;
    END LOOP;
END;
/
```

Execute the code from the `lab_02_01.sql` file. **It will show `FALSE - 105` for each row fetched.**

2. Modify the following snippet of code to make better use of the `FOR UPDATE` clause and improve the performance of the program.

```
DECLARE
  CURSOR cur_update
    IS SELECT * FROM customers
    WHERE credit_limit < 5000 FOR UPDATE;
BEGIN
  FOR v_rec IN cur_update
    LOOP
      IF v_rec IS NOT NULL THEN
        UPDATE customers
          SET credit_limit = credit_limit + 200
          WHERE customer_id = v_rec.customer_id;
      END IF;
    END LOOP;
END;
/
```

Oracle University and En-Sof Informatica E Treinamento Ltda use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 2: Solutions (continued)

Modify the file `lab_02_02.sql` file as shown below:

```
DECLARE
  CURSOR cur_update
  IS SELECT * FROM customers
  WHERE credit_limit < 5000 FOR UPDATE;
BEGIN
  FOR v_rec IN cur_update
    LOOP
      UPDATE customers
      SET credit_limit = credit_limit + 200
      WHERE CURRENT OF cur_update;
    END LOOP;
END;
/
```

Alternatively, you can execute the code from the `sol_02_02.sql` file.

1. Create a package specification that defines subtypes, which can be used for the `warranty_period` field of the product_information table. Name this package `MY_TYPES`. The type needs to hold the month and year for a warranty period.

```
CREATE OR REPLACE PACKAGE mytypes
IS
  TYPE typ_warranty
    IS RECORD (month POSITIVE, year PLS_INTEGER);
  SUBTYPE warranty IS typ_warranty; -- based on RECORD type
END mytypes;
/
```

4. Create a package named `SHOW_DETAILS` that contains two subroutines. The first subroutine should show order details for the given `order_id`. The second subroutine should show customer details for the given `customer_id`, including the customer Id, first name, phone numbers, credit limit, and email address.

   Both the subroutines should use the cursor variable to return the necessary details.

```
CREATE OR REPLACE PACKAGE show_details AS

TYPE rt_order IS REF CURSOR RETURN orders%ROWTYPE;

TYPE typ_cust_rec IS RECORD
  (cust_id NUMBER(6), cust_name VARCHAR2(20),
   custphone customers.phone_numbers%TYPE,
   credit NUMBER(9,2), cust_email VARCHAR2(30));
TYPE rt_cust IS REF CURSOR RETURN typ_cust_rec;

PROCEDURE get_order(p_orderid IN NUMBER, p_cv_order IN OUT rt_order);
```

Oracle University and En-Sof Informatica E Treinamento Ltda use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 2: Solutions (continued)

```
PROCEDURE get_cust(p_custid IN NUMBER, p_cv_cust IN OUT rt_cust);
END show_details;
/

CREATE OR REPLACE PACKAGE BODY show_details AS
PROCEDURE get_order
  (p_orderid IN NUMBER, p_cv_order IN OUT rt_order)
IS
BEGIN
  OPEN p_cv_order FOR
    SELECT * FROM order
      WHERE order_id = p_orderid;
-- CLOSE p_cv_order
END;

PROCEDURE get_cust
  (p_custid IN NUMBER, p_cv_cust IN OUT rt_cust)
IS
BEGIN
  OPEN p_cv_cust FOR
    SELECT customer_id, cust_first_name, phone_numbers, credit_limit,
           cust_email FROM customers
      WHERE customer_id = p_custid;
-- CLOSE p_cv_cust
END;
END;
/
```

Alternatively, you can execute the code from the `sol_02_04.sql` file.

Oracle University and En-Sof Informatica E Treinamento Ltda use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 3: Solutions

Collection Analysis

1. Examine the following definitions. Run the `lab_03_01.sql` script to create these objects.

```
CREATE TYPE typ_item AS OBJECT  --create object
 (prodid  NUMBER(5),
  price   NUMBER(7,2) )
/
CREATE TYPE typ_item_nst -- define nested table type
  AS TABLE OF typ_item
/
CREATE TABLE POrder (  -- create database table
     ordid NUMBER(5),
     supplier   NUMBER(5),
     requester  NUMBER(4),
     ordered    DATE,
     items typ_item_nst)
     NESTED TABLE items STORE AS item_stor_tab
/
@lab_03_01
```

2. The code shown below generates an error. Run the `lab_03_02.sql` script to generate and view the error.

```
BEGIN
  -- Insert an order
  INSERT INTO pOrder
    (ordid, supplier, requester, ordered, items)
    VALUES (1000, 12345, 9876, SYSDATE, NULL);
  -- insert the items for the order created
  INSERT INTO TABLE (SELECT items
                     FROM   pOrder
                     WHERE  ordid = 1000)
    VALUES(typ_item(99, 129.00));
END;
/
@lab_03_02
```

Why is the error occurring?

The error: ORA-22908: reference to NULL table value is resulting from setting the table columns to NULL.

Oracle University and En-Sof Informatica E Treinamento Ltda use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 3: Solutions (continued)

How can you fix the error?

Always use a nested table's default constructor to initialize it:

```
TRUNCATE TABLE pOrder;

-- A better approach is to avoid setting the table
-- column to NULL, and instead, use a nested table's
-- default constructor to initialize
BEGIN
  -- Insert an order
  INSERT INTO pOrder
    (ordid, supplier, requester, ordered, items)
    VALUES (1000, 12345, 9876, SYSDATE,
            typ_item_nst(typ_item(99, 129.00)));
END;
/

-- However, if the nested table is set to NULL, you can
-- use an UPDATE statement to set its value.
BEGIN
  -- Insert an order
  INSERT INTO pOrder
    (ordid, supplier, requester, ordered, items)
    VALUES (1000, 12345, 9876, SYSDATE, null);
  -- Once the nested table is set to null, use the update
  -- update statement
  UPDATE pOrder
    SET items = typ_item_nst(typ_item(99, 129.00))
    WHERE ordid = 1000
END;
/
```

Oracle University and En-Sof Informatica E Treinamento Ltda use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 3: Solutions (continued)

3. Examine the following code. This code produces an error. Which line causes the error, and how do you fix it? (**Note:** You can run the `lab_03_03.sql` script to view the error output).

```
DECLARE
  TYPE credit_card_typ
  IS VARRAY(100) OF VARCHAR2(30);
  v_mc   credit_card_typ := credit_card_typ();
  v_visa credit_card_typ := credit_card_typ();
  v_am   credit_card_typ;
  v_disc credit_card_typ := credit_card_typ();
  v_dc   credit_card_typ := credit_card_typ();
BEGIN
  v_mc.EXTEND;
  v_visa.EXTEND;
  v_am.EXTEND;
  v_disc.EXTEND;
  v_dc.EXTEND;
END;
/
```

This causes an ORA-06531: Reference to uninitialized collection. To fix it, initialize the `v_am` variable by using the same technique as the others:

```
DECLARE
  TYPE credit_card_typ
  IS VARRAY(100) OF VARCHAR2(30);

  v_mc   credit_card_typ := credit_card_typ();
  v_visa credit_card_typ := credit_card_typ();
  v_am   credit_card_typ := credit_card_typ();
  v_disc credit_card_typ := credit_card_typ();
  v_dc   credit_card_typ := credit_card_typ();

BEGIN
  v_mc.EXTEND;
  v_visa.EXTEND;
  v_am.EXTEND;
  v_disc.EXTEND;
  v_dc.EXTEND;
END;
/
```

Oracle University and En-Sof Informatica E Treinamento Ltda use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 3: Solutions (continued)

In the following practice exercises, you will implement a nested table column in the CUSTOMERS table and write PL/SQL code to manipulate the nested table.

4. Create a nested table to hold credit card information.

   Create an object type called typ_cr_card. It should have the following specification:

```
card_type  VARCHAR2(25)
card_num   NUMBER
```

   Create a nested table type called typ_cr_card_nst that is a table of typ_cr_card.

```
CREATE TYPE typ_cr_card AS OBJECT  --create object
 (card_type  VARCHAR2(25),
  card_num   NUMBER);
/
CREATE TYPE typ_cr_card_nst -- define nested table type
  AS TABLE OF typ_cr_card;
/
```

   Add a column to the CUSTOMERS table called credit_cards. Make this column a nested table of type typ_cr_card_nst. You can use the following syntax:

```
ALTER TABLE customers ADD
  credit_cards typ_cr_card_nst
   NESTED TABLE credit_cards STORE AS c_c_store_tab;
```

5. Create a PL/SQL package that manipulates the credit_cards column in the CUSTOMERS table.

   Open the lab_03_05.sql file. It contains the package specification and part of the package body. Complete the code so that the package:

   - Inserts credit card information (the credit card name and number for a specific customer.)
   - Displays credit card information in an unnested format.

```
CREATE OR REPLACE PACKAGE credit_card_pkg
IS
  PROCEDURE update_card_info
    (p_cust_id NUMBER, p_card_type VARCHAR2, p_card_no VARCHAR2);

  PROCEDURE display_card_info
    (p_cust_id NUMBER);
END credit_card_pkg;  -- package spec
/
```

Oracle University and En-Sof Informatica E Treinamento Ltda use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 3: Solutions (continued)

```
CREATE OR REPLACE PACKAGE BODY credit_card_pkg
IS
  PROCEDURE update_card_info
    (p_cust_id NUMBER, p_card_type VARCHAR2, p_card_no VARCHAR2)
  IS
    v_card_info typ_cr_card_nst;
    i INTEGER;
  BEGIN
    SELECT credit_cards
      INTO v_card_info
      FROM customers
      WHERE customer_id = p_cust_id;
    IF v_card_info.EXISTS(1) THEN  -- cards exist, add more
      i := v_card_info.LAST;
      v_card_info.EXTEND(1);
      v_card_info(i+1) := typ_cr_card(p_card_type, p_card_no);
      UPDATE customers
        SET  credit_cards = v_card_info
        WHERE customer_id = p_cust_id;
    ELSE   -- no cards for this customer yet, construct one
      UPDATE customers
        SET  credit_cards = typ_cr_card_nst
            (typ_cr_card(p_card_type, p_card_no))
        WHERE customer_id = p_cust_id;
    END IF;
  END update_card_info;

  PROCEDURE display_card_info
    (p_cust_id NUMBER)
  IS
    v_card_info typ_cr_card_nst;
    i INTEGER;
  BEGIN
    SELECT credit_cards
      INTO v_card_info
      FROM customers
      WHERE customer_id = p_cust_id;
    IF v_card_info.EXISTS(1) THEN
      FOR idx IN v_card_info.FIRST..v_card_info.LAST LOOP
     DBMS_OUTPUT.PUT('Card Type: ' || v_card_info(idx).card_type || ' ');
     DBMS_OUTPUT.PUT_LINE('/ Card No: ' || v_card_info(idx).card_num );
      END LOOP;
    ELSE
      DBMS_OUTPUT.PUT_LINE('Customer has no credit cards.');
    END IF;
  END display_card_info;
END credit_card_pkg;  -- package body
/
```

Oracle University and En-Sof Informatica E Treinamento Ltda use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

6. Test your package with the following statements and output:

```
SET SERVEROUT ON

EXECUTE credit_card_pkg.display_card_info(120)
Customer has no credit cards.
PL/SQL procedure successfully completed.

EXECUTE credit_card_pkg.update_card_info -
    (120, 'Visa', 11111111)
PL/SQL procedure successfully completed.

SELECT credit_cards
FROM   customers
WHERE  customer_id = 120;

CREDIT_CARDS(CARD_TYPE, CARD_NUM)
-------------------------------------------------
TYP_CR_CARD_NST(TYP_CR_CARD('Visa', 11111111))

EXECUTE credit_card_pkg.display_card_info(120)
Card Type: Visa / Card No: 11111111
PL/SQL procedure successfully completed.

EXECUTE credit_card_pkg.update_card_info -
    (120, 'MC', 2323232323)
PL/SQL procedure successfully completed.

EXECUTE credit_card_pkg.update_card_info -
    (120, 'DC', 4444444)
PL/SQL procedure successfully completed.

EXECUTE credit_card_pkg.display_card_info(120)
Card Type: Visa / Card No: 11111111
Card Type: MC / Card No: 2323232323
Card Type: DC / Card No: 4444444
PL/SQL procedure successfully completed.
```

Oracle University and En-Sof Informatica E Treinamento Ltda use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 3: Solutions (continued)

7. Write a SELECT statement against the CREDIT_CARDS column to unnest the data. Use the TABLE expression.

   For example, if the SELECT statement returns:

```
SELECT credit_cards
FROM   customers
WHERE  customer_id = 120;

CREDIT_CARDS(CARD_TYPE, CARD_NUM)
-------------------------------------------------------
TYP_CR_CARD_NST(TYP_CR_CARD('Visa', 11111111), TYP_CR_CARD('MC',
2323232323), TYP_CR_CARD('DC', 4444444))

-- Use the table expression so that the result is:

CUSTOMER_ID CUST_LAST_NAME  CARD_TYPE        CARD_NUM
----------- --------------- ------------- -----------
        120 Higgins         Visa             11111111
        120 Higgins         MC             2323232323
        120 Higgins         DC                4444444
```

```
SELECT c1.customer_id, c1.cust_last_name, c2.*
FROM   customers c1, TABLE(c1.credit_cards) c2
WHERE  customer_id = 120;
```

Oracle University and En-Sof Informatica E Treinamento Ltda use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 4: Solutions

1. An external C routine definition is created for you. The `.c` file is stored in the `$HOME/labs` directory on the database server. This function returns the tax amount based on the total sales figure passed to it as a parameter. The name of the `.c` file is named as `calc_tax.c`. The shared object filename is `calc_tax.so`. The function is defined as:

```
calc_tax(n)
  int n;
  {
     int tax;
     tax=(n*8)/100;
     return(tax);
  }
```

   a. Create a `calc_tax.so` file using the following command:

```
cc -shared -o calc_tax.so calc_tax.c
```

   b. Copy the file `calc_tax.so` to `$ORACLE_HOME/bin` directory using the following command:

```
cp calc_tax.so $ORACLE_HOME/bin
```

   c. Log in to SQL*Plus. Create the library object. Name the library object `c_code` and define its path as:

```
CREATE OR REPLACE LIBRARY c_code AS '$ORACLE_HOME/bin/calc_tax.so';
```

   d. Create a function named `call_c` to publish the external C routine. This fuction has one numeric parameter and it returns a binary integer. Identify the `AS LANGUAGE`, `LIBRARY`, and `NAME` clauses of the function.

```
CREATE OR REPLACE FUNCTION call_c
(x BINARY_INTEGER)
RETURN BINARY_INTEGER
AS LANGUAGE C
LIBRARY c_code
NAME "calc_tax";
/
```

Oracle University and En-Sof Informatica E Treinamento Ltda use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 4: Solutions (continued)

e. Create a procedure to call the `call_c` function created in the the previous step. Name this procedure `c_output`. It has one numeric parameter. Include a `DBMS_OUTPUT.PUT_LINE` statement so that you can view the results returned from your C function.

```
CREATE OR REPLACE PROCEDURE c_output
  (p_in IN BINARY_INTEGER)
IS
  i BINARY_INTEGER;
BEGIN
  i := call_c(p_in);
  DBMS_OUTPUT.PUT_LINE('The total tax is: ' || i);
END c_output;
/
```

f. Set the serveroutput ON.

```
SET SERVEROUTPUT ON
```

g. Execute the `c_output` procedure.

```
EXECUTE c_output(1000000)
The total tax is: 8000

PL/SQL procedure successfully completed.
```

Oracle University and En-Sof Informatica E Treinamento Ltda use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 4: Solutions (continued)

2. A Java method definition is created for you. The method accepts a 16-digit credit card number as the argument and returns the formatted credit card number (4 digits followed by a space). The `.java` file is stored in your `$HOME/labs` directory. The name of the .class file is `FormatCreditCardNo.class`. The method is defined as:

```
public class FormatCreditCardNo
{
  public static final void formatCard(String[] cardno)
  {
  int count=0, space=0;
  String oldcc=cardno[0];
  String[] newcc= {""};
  while (count<16)
  {
  newcc[0]+= oldcc.charAt(count);
  space++;
  if (space ==4)
  {  newcc[0]+=" "; space=0;  }
  count++;
  }
  cardno[0]=newcc [0];
  }
}
```

a. Load the `.java` source file. From the operating system, type:

```
loadjava -user oe/oe FormatCreditCardNo.java
```

b. Publish the Java class method by defining a PL/SQL procedure named `CCFORMAT`. This procedure accepts one `IN OUT` parameter.

Use the following definition for the `NAME` parameter:

```
NAME 'FormatCreditCardNo.formatCard(java.lang.String[])';
```

Create a file named `ccformat.sql` and enter the following code.

```
CREATE OR REPLACE PROCEDURE ccformat
(x IN OUT VARCHAR2)
AS LANGUAGE JAVA
NAME 'FormatCreditCardNo.formatCard(java.lang.String[])';
/
```

Save `ccformat.sql`.

Oracle University and En-Sof Informatica E Treinamento Ltda use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 4: Solutions (continued)

c. Execute the Java class method. Define a SQL*Plus variable, intialize it, run the `ccformat.sql` file and use the execute command to execute the `ccformat` procedure. Finally, print the SQL*Plus variable.

```
VARIABLE x VARCHAR2(20)
EXECUTE :x := '1234567887654321'
PL/SQL procedure successfully completed.

@ccformat
Procedure created.

EXECUTE ccformat(:x)
PL/SQL procedure successfully completed.

PRINT x
X
----------------------------------------
1234 5678 8765 4321
```

Oracle University and En-Sof Informatica E Treinamento Ltda use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 5: Solutions

1. Create a PL/SQL server page to display order information. The name of the procedure that you are creating is `show_orders`.

   a. Open the `lab_05_01.psp` file. This file contains some HTML code.

   b. At the top of the file, include these directives:

```
<%@ page language="PL/SQL" %>
<%@ plsql procedure="show_orders" %>
```

   c. Use the following SQL statement to retrieve the order details. Place the following statement in the `FOR` loop:

```
SELECT order_id, order_mode, customer_id, order_status,
       order_total, call_c(order_total) tax, sales_rep_id
FROM   orders
ORDER BY order_id;
```

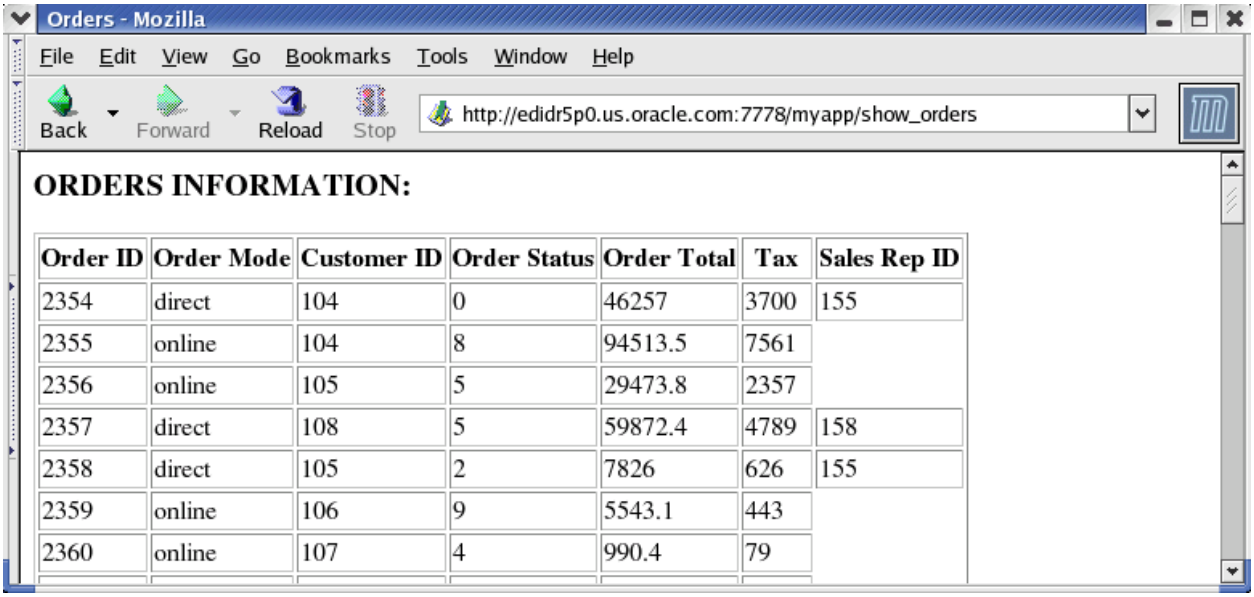
   d. Load the PSP file from `$HOME/labs`, type:

```
loadpsp -replace -user oe/oe lab_05_01.psp
```

   Note: If the HTTP Server is not started, please start it using the following command:

```
opmnctl startall
```

   e. From your browser, request the `show_orders` PSP as shown below.



Open `sol_05_01.psp` to see the modified code.

Oracle University and En-Sof Informatica E Treinamento Ltda use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

2. Create a PL/SQL server page to display the following customer information:

   ```
   CUSTOMER_ID
   CUST_FIRST_NAME
   CUST_LAST_NAME
   CREDIT_LIMIT
   CUST_EMAIL
   ```

   The name of the procedure is `SHOW_CUST` and you need to pass the `CUSTOMER_ID` as the parameter.

   a. Open the `lab_05_02a.psp` file. This file contains some HTML code.

   b. At the top of the file, include these directives:

```
<%@ page language="PL/SQL" %>
<%@ plsql procedure="show_cust" %>
<%@ plsql parameter="custid" %> type="NUMBER" default="101" %>
```

   c. Place the parameter as shown in the following command:

```
<p>Following are the details for the Customer ID <%= custid %>
```

   d. Use the following SQL statement to retrieve customer information. Place this statement in the `FOR` loop within the `lab_05_02a.psp` file.

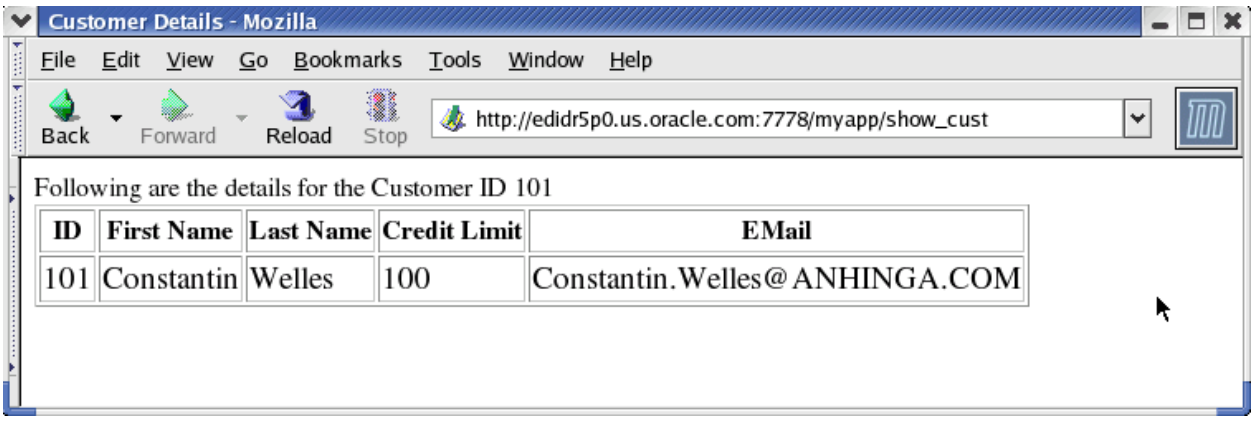
```
SELECT * FROM customers WHERE customer_id = custid;
```

Note: You can access the `sol_05_02a.psp` file for the modified code.

   e. Load the PSP file from `$HOME/labs`, type:

```
loadpsp -replace -user oe/oe lab_05_02a.psp
```

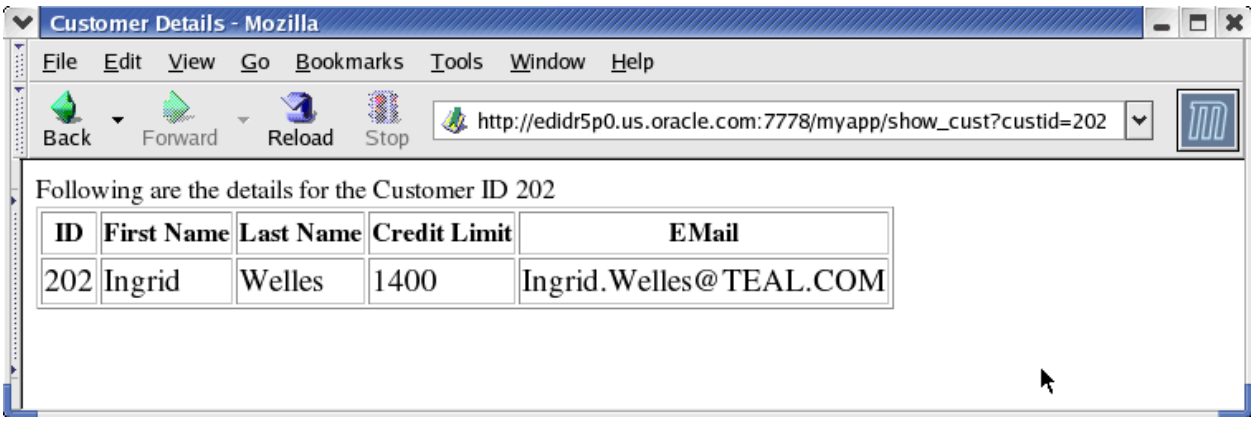
   f. From the browser, request the `show_cust` PSP. By default it will show details for `CUSTOMER_ID 101` because that is the specified default value.



THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and En-Sof Informatica E Treinamento Ltda use only

g. To see details for other customers, pass the parameter as shown below.



h. To create an HTML form for calling the PSP, open `lab_05_02b.psp` and add the highlighted details.

```
<%@ page language="PL/SQL" %>
<%@ plsql procedure="show_cust_call" %>
<%@ plsql parameter="custid" %> type="NUMBER" default="101" %>
<HTML>
<BODY>
<form method="POST" action="show_cust">
<p>Enter the Customer ID:
<input type="text" name="custid">
<input type="submit" value="Submit">
</form>
</BODY>
</HTML>
```

i. Save the PSP file.

Note: You can access the `sol_05_02b.psp` file for the modified code.

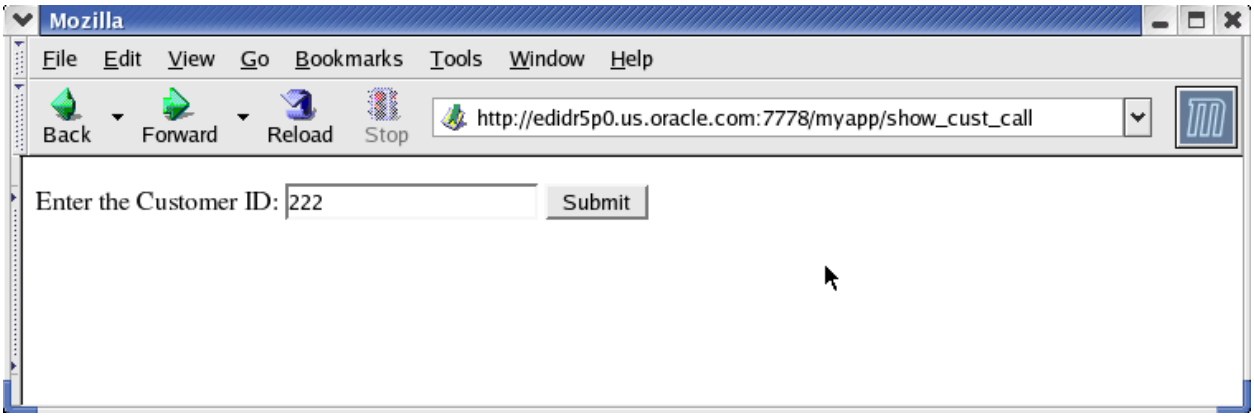
j. Load the PSP file from `$HOME/labs`, and enter:

```
loadpsp -replace -user oe/oe lab_05_02b.psp
```

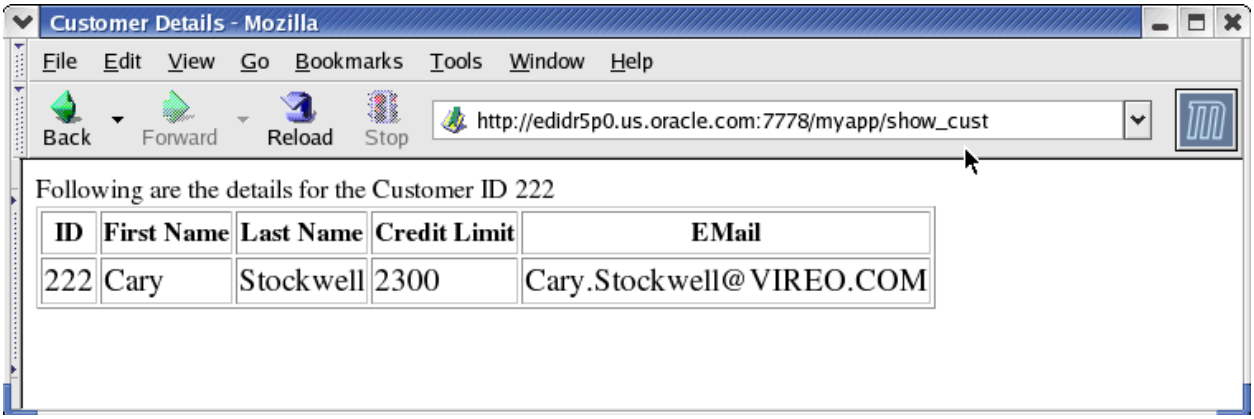
Oracle University and En-Sof Informatica E Treinamento Ltda use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

k. From the browser, request `show_cust_call` PSP. Enter the Customer ID and click the Submit button.



l. Note that the form in turn calls the `show_cust` PSP and then displays the details.



Oracle University and En-Sof Informatica E Treinamento Ltda use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 6: Solutions

In this practice you will define an application context and security policy to implement the policy: "Sales Representatives can see their own order information only in the ORDERS table." You will create sales representative IDs to test the success of your implementation.

Examine the definition of the ORDERS table, and the sales representative's data:

1. Examine, then run the lab_06_01.sql script.

   This script will create the sales representative's ID accounts with appropriate privileges to access the database:

```
CONNECT /AS sysdba

CREATE USER sr153 IDENTIFIED BY oracle
 DEFAULT TABLESPACE USERS
 TEMPORARY TABLESPACE TEMP
 QUOTA UNLIMITED ON USERS;

CREATE USER sr154 IDENTIFIED BY oracle
 DEFAULT TABLESPACE USERS
 TEMPORARY TABLESPACE TEMP
 QUOTA UNLIMITED ON USERS;

...

CREATE USER sr163 IDENTIFIED BY oracle
 DEFAULT TABLESPACE USERS
 TEMPORARY TABLESPACE TEMP
 QUOTA UNLIMITED ON USERS;

GRANT create session
    , alter session
   TO sr153, sr154, sr155, sr156, sr158, sr159,
   sr160, sr161, sr163;
GRANT SELECT, INSERT, UPDATE, DELETE ON
  oe.orders TO sr153, sr154, sr155, sr156, sr158,
                 sr159, sr160, sr161, sr163;
GRANT SELECT, INSERT, UPDATE, DELETE ON
  oe.order_items TO sr153, sr154, sr155, sr156, sr158,
                 sr159, sr160, sr161, sr163;

CREATE PUBLIC SYNONYM orders FOR oe.orders;
CREATE PUBLIC SYNONYM orders FOR oe.order_items;

CONNECT oe/oe
```

```
@lab_06_01.sql
```

Oracle University and En-Sof Informatica E Treinamento Ltda use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

2. Set up an application context:

   Connect to the database as `SYSDBA` before creating this context.

   Create an application context named `sales_orders_ctx`.

   Associate this context to the `oe.sales_orders_pkg`.

```
CONNECT /AS sysdba

CREATE CONTEXT sales_orders_ctx
USING oe.sales_orders_pkg;
```

3. Connect as `OE/OE`.

   Examine this package specification:

```
CREATE OR REPLACE PACKAGE sales_orders_pkg
IS
 PROCEDURE set_app_context;
 FUNCTION the_predicate
  (p_schema VARCHAR2, p_name VARCHAR2)
   RETURN VARCHAR2;
END sales_orders_pkg;    -- package spec
/
```

   Create this package specification and then the package body in the `OE` schema.

   When you create the package body, set up two constants as follows:

```
c_context CONSTANT VARCHAR2(30) := 'SALES_ORDER_CTX';
c_attrib  CONSTANT VARCHAR2(30) := 'SALES_REP';
```

   Use these constants in the `SET_APP_CONTEXT` procedure to set the application context to the current user.

Oracle University and En-Sof Informatica E Treinamento Ltda use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 6: Solutions (continued)

```
CREATE OR REPLACE PACKAGE BODY sales_orders_pkg
IS
  c_context CONSTANT VARCHAR2(30) := 'SALES_ORDERS_CTX';
  c_attrib  CONSTANT VARCHAR2(30) := 'SALES_REP';

PROCEDURE set_app_context
  IS
    v_user VARCHAR2(30);
BEGIN
  SELECT user INTO v_user FROM dual;
  DBMS_SESSION.SET_CONTEXT
    (c_context, c_attrib, v_user);
END set_app_context;

FUNCTION the_predicate
(p_schema VARCHAR2, p_name VARCHAR2)
RETURN VARCHAR2
IS
  v_context_value VARCHAR2(100) :=
     SYS_CONTEXT(c_context, c_attrib);
  v_restriction VARCHAR2(2000);
BEGIN
  IF v_context_value LIKE 'SR%'  THEN
    v_restriction :=
     'SALES_REP_ID =
      SUBSTR(''' || v_context_value || ''', 3, 3)';
  ELSE
    v_restriction := null;
  END IF;
  RETURN v_restriction;
END the_predicate;

END sales_orders_pkg; -- package body
/
```

Oracle University and En-Sof Informatica E Treinamento Ltda use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 6: Solutions (continued)

4. Connect as SYSDBA and define the policy.

   Use DBMS_RLS.ADD_POLICY to define the policy.

   Use these specifications for the parameter values:

```
object_schema   OE
object_name     ORDERS
policy_name     OE_ORDERS_ACCESS_POLICY
function_schema OE
policy_function SALES_ORDERS_PKG.THE_PREDICATE
statement_types SELECT, INSERT, UPDATE, DELETE
update_check    FALSE,
enable          TRUE);
```

```
CONNECT /as sysdba

DECLARE
BEGIN
  DBMS_RLS.ADD_POLICY (
   'OE',
   'ORDERS',
   'OE_ORDERS_ACCESS_POLICY',
   'OE',
   'SALES_ORDERS_PKG.THE_PREDICATE',
   'SELECT, INSERT, UPDATE, DELETE',
   FALSE,
   TRUE);
END;
/
```

5. Connect as SYSDBA and create a logon trigger to implement fine-grained access control. You can call the trigger SET_ID_ON_LOGON. This trigger causes the context to be set as each user is logged on.

```
CONNECT /as sysdba

CREATE OR REPLACE TRIGGER set_id_on_logon
AFTER logon on DATABASE
BEGIN
  oe.sales_orders_pkg.set_app_context;
END;
/
```

Oracle University and En-Sof Informatica E Treinamento Ltda use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 6: Solutions (continued)

6. Test the fine-grained access implementation. Connect as your SR user and query the ORDERS table. For example, your results should match:

```
CONNECT sr153/oracle

SELECT sales_rep_id, COUNT(*)
FROM   orders
GROUP BY sales_rep_id;

SALES_REP_ID   COUNT(*)
------------ ----------
         153          5

CONNECT sr154/oracle

SELECT sales_rep_id, COUNT(*)
FROM   orders
GROUP BY sales_rep_id;

SALES_REP_ID   COUNT(*)
------------ ----------
         154         10
```

Note

During debugging, you may need to disable or remove some of the objects created for this lesson.

If you need to disable the logon trigger, issue the command:

```
ALTER TRIGGER set_id_on_logon DISABLE;
```

If you need to remove the policy you created, issue the command:

```
EXECUTE DBMS_RLS.DROP_POLICY('OE', 'ORDERS',-
   'OE_ORDERS_ACCESS_POLICY')
```

Oracle University and En-Sof Informatica E Treinamento Ltda use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 7: Solutions

1. In this exercise, you will pin the fine-grained access package created in Lesson 6.

   Note: If you have not completed practice 6, run the following files in the `$HOME/soln` folder:

```
@sol_06_02.sql
@sol_06_03.sql
@sol_06_04.sql
@sol_06_05.sql
```

   Using the `DBMS_SHARED_POOL.KEEP` procedure, pin your `SALES_ORDERS_PKG`.

```
EXECUTE sys.dbms_shared_pool.keep('SALES_ORDERS_PKG')
```

   Execute the `DBMS_SHARED_POOL.SIZES` procedure to see the objects in the shared pool that are larger than 500 kilobytes.

```
SET SERVEROUTPUT ON
EXECUTE sys.dbms_shared_pool.sizes(500)
```

2. Open the `lab_07_02.sql` file and examine the package (the package body is shown below):

```
CREATE OR REPLACE PACKAGE BODY credit_card_pkg
IS
  PROCEDURE update_card_info
    (p_cust_id NUMBER, p_card_type VARCHAR2, p_card_no VARCHAR2)
  IS
    v_card_info typ_cr_card_nst;
    i INTEGER;
  BEGIN
    SELECT credit_cards
      INTO v_card_info
      FROM customers
      WHERE customer_id = p_cust_id;
    IF v_card_info.EXISTS(1) THEN  -- cards exist, add more
      i := v_card_info.LAST;
      v_card_info.EXTEND(1);
      v_card_info(i+1) := typ_cr_card(p_card_type, p_card_no);
      UPDATE customers
        SET  credit_cards = v_card_info
        WHERE customer_id = p_cust_id;
    ELSE   -- no cards for this customer yet, construct one
      UPDATE customers
        SET  credit_cards = typ_cr_card_nst
            (typ_cr_card(p_card_type, p_card_no))
        WHERE customer_id = p_cust_id;
    END IF;
  END update_card_info;
-- continued on next page
```

Oracle University and En-Sof Informatica E Treinamento Ltda use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 7: Solutions (continued)

```
-- continued from previous page.
  PROCEDURE display_card_info
    (p_cust_id NUMBER)
  IS
    v_card_info typ_cr_card_nst;
    i INTEGER;
  BEGIN
    SELECT credit_cards
      INTO v_card_info
      FROM customers
      WHERE customer_id = p_cust_id;
    IF v_card_info.EXISTS(1) THEN
      FOR idx IN v_card_info.FIRST..v_card_info.LAST LOOP
          DBMS_OUTPUT.PUT('Card Type: ' || v_card_info(idx).card_type
           || ' ');
        DBMS_OUTPUT.PUT_LINE('/ Card No: ' ||
           v_card_info(idx).card_num );
      END LOOP;
    ELSE
      DBMS_OUTPUT.PUT_LINE('Customer has no credit cards.');
    END IF;
  END display_card_info;
END credit_card_pkg;  -- package body
/
```

This code needs to be improved. The following issues exist in the code:

- The local variables use the `INTEGER` data type.
- The same `SELECT` statement is run in the two procedures.
- The same `IF v_card_info.EXISTS(1) THEN` statement is in the two procedures.

Oracle University and En-Sof Informatica E Treinamento Ltda use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 7: Solutions (continued)

3. To improve the code, make the following modifications:

   Change the local INTEGER variables to use a more efficient data type.

   Move the duplicated code into a function. The package specification for the modification is:

```
CREATE OR REPLACE PACKAGE credit_card_pkg
IS
  FUNCTION cust_card_info
    (p_cust_id NUMBER, p_card_info IN OUT typ_cr_card_nst )
    RETURN BOOLEAN;
  PROCEDURE update_card_info
    (p_cust_id NUMBER, p_card_type VARCHAR2, p_card_no VARCHAR2);
   PROCEDURE display_card_info
    (p_cust_id NUMBER);
END credit_card_pkg;  -- package spec
/
```

   Have the function return TRUE if the customer has credit cards. The function should return FALSE if the customer does not have credit cards. Pass into the function an uninitialized nested table. The function places the credit card information into this uninitialized parameter.

```
CREATE OR REPLACE PACKAGE credit_card_pkg
IS
  FUNCTION cust_card_info
    (p_cust_id NUMBER, p_card_info IN OUT typ_cr_card_nst )
    RETURN BOOLEAN;

  PROCEDURE update_card_info
    (p_cust_id NUMBER, p_card_type VARCHAR2, p_card_no VARCHAR2);

  PROCEDURE display_card_info
    (p_cust_id NUMBER);

END credit_card_pkg;  -- package spec
/

-- continued on next page
```

Oracle University and En-Sof Informatica E Treinamento Ltda use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 7: Solutions (continued)

```
CREATE OR REPLACE PACKAGE BODY credit_card_pkg
IS
  FUNCTION cust_card_info
    (p_cust_id NUMBER, p_card_info IN OUT typ_cr_card_nst )
    RETURN BOOLEAN
  IS
    v_card_info_exists BOOLEAN;
  BEGIN
    SELECT credit_cards
      INTO p_card_info
      FROM customers
      WHERE customer_id = p_cust_id;
    IF p_card_info.EXISTS(1) THEN
      v_card_info_exists := TRUE;
    ELSE
      v_card_info_exists := FALSE;
    END IF;
    RETURN v_card_info_exists;
  END cust_card_info;

  PROCEDURE update_card_info
    (p_cust_id NUMBER, p_card_type VARCHAR2, p_card_no VARCHAR2)
  IS
    v_card_info typ_cr_card_nst;
    i PLS_INTEGER;
  BEGIN
    IF cust_card_info(p_cust_id, v_card_info) THEN
      -- cards exist, add more
      i := v_card_info.LAST;
      v_card_info.EXTEND(1);
      v_card_info(i+1) := typ_cr_card(p_card_type, p_card_no);
      UPDATE customers
        SET  credit_cards = v_card_info
        WHERE customer_id = p_cust_id;
    ELSE   -- no cards for this customer yet, construct one
      UPDATE customers
        SET  credit_cards = typ_cr_card_nst
            (typ_cr_card(p_card_type, p_card_no))
        WHERE customer_id = p_cust_id;
    END IF;
  END update_card_info;

-- continued on next page
```

Oracle University and En-Sof Informatica E Treinamento Ltda use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

```
PROCEDURE display_card_info
    (p_cust_id NUMBER)
  IS
    v_card_info typ_cr_card_nst;
    i PLS_INTEGER;
  BEGIN
    IF cust_card_info(p_cust_id, v_card_info) THEN
      FOR idx IN v_card_info.FIRST..v_card_info.LAST LOOP
          DBMS_OUTPUT.PUT('Card Type: ' ||
          v_card_info(idx).card_type || ' ');
        DBMS_OUTPUT.PUT_LINE('/ Card No: ' ||
          v_card_info(idx).card_num );
      END LOOP;
    ELSE
      DBMS_OUTPUT.PUT_LINE('Customer has no credit cards.');
    END IF;
  END display_card_info;
END credit_card_pkg;  -- package body
/
```

4. Test your modified code with the following data:

```
EXECUTE credit_card_pkg.update_card_info -
        (120, 'AM EX', 55555555555)
PL/SQL procedure successfully completed.

EXECUTE credit_card_pkg.display_card_info(120)
Card Type: Visa / Card No: 11111111
Card Type: MC / Card No: 2323232323
Card Type: DC / Card No: 4444444
Card Type: AM EX / Card No: 55555555555

PL/SQL procedure successfully completed.
-- Note: If you did not complete Practice 3, your results
-- will be:
EXECUTE credit_card_pkg.display_card_info(120)
Card Type: AM EX / Card No: 55555555555

PL/SQL procedure successfully completed.
```

Oracle University and En-Sof Informatica E Treinamento Ltda use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 7: Solutions (continued)

5. Open the `lab_07_05a.sql` file. It contains the modified code from the previous question #3.

   You need to modify the `UPDATE_CARD_INFO` procedure to return information (using the `RETURNING` clause) about the credit cards being updated. Assume that this information will be used by another application developer in your team, who is writing a graphical reporting utility on customer credit cards, after a customer's credit card information is changed.

   Modify the code to use the `RETURNING` clause to find information about the row affected by the `UPDATE` statements.

```
CREATE OR REPLACE PACKAGE credit_card_pkg
IS
  FUNCTION cust_card_info
    (p_cust_id NUMBER, p_card_info IN OUT typ_cr_card_nst )
    RETURN BOOLEAN;

  PROCEDURE update_card_info
    (p_cust_id NUMBER, p_card_type VARCHAR2,
     p_card_no VARCHAR2, o_card_info OUT typ_cr_card_nst);

  PROCEDURE display_card_info
    (p_cust_id NUMBER);

END credit_card_pkg;  -- package spec
/
```

Oracle University and En-Sof Informatica E Treinamento Ltda use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 7: Solutions (continued)

```
... only the update_card_info procedure is changed in the body

PROCEDURE update_card_info
    (p_cust_id NUMBER, p_card_type VARCHAR2,
     p_card_no VARCHAR2, o_card_info OUT typ_cr_card_nst)
  IS
    v_card_info typ_cr_card_nst;
    i PLS_INTEGER;
  BEGIN
    IF cust_card_info(p_cust_id, v_card_info) THEN
      -- cards exist, add more
      i := v_card_info.LAST;
      v_card_info.EXTEND(1);
      v_card_info(i+1) := typ_cr_card(p_card_type, p_card_no);
      UPDATE customers
        SET  credit_cards = v_card_info
        WHERE customer_id = p_cust_id
        RETURNING credit_cards INTO o_card_info;
    ELSE   -- no cards for this customer yet, construct one
      UPDATE customers
        SET  credit_cards = typ_cr_card_nst
            (typ_cr_card(p_card_type, p_card_no))
        WHERE customer_id = p_cust_id
        RETURNING credit_cards INTO o_card_info;
    END IF;
END update_card_info;
...
```

You can test your modified code with the following procedure (contained in `lab_07_05b.sql`):

```
CREATE OR REPLACE PROCEDURE test_credit_update_info
(p_cust_id NUMBER, p_card_type VARCHAR2, p_card_no NUMBER)
IS
  v_card_info typ_cr_card_nst;
BEGIN
  credit_card_pkg.update_card_info
    (p_cust_id, p_card_type, p_card_no, v_card_info);
END test_credit_update_info;
/
```

Oracle University and En-Sof Informatica E Treinamento Ltda use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Test your code with the following statements set in boldface:

```
EXECUTE test_credit_update_info(125, 'AM EX', 123456789)
PL/SQL procedure successfully completed.
SELECT credit_cards FROM customers WHERE customer_id = 125;
CREDIT_CARDS(CARD_TYPE, CARD_NUM)
------------------------------------------------------------
TYP_CR_CARD_NST(TYP_CR_CARD('AM EX', 123456789))
```

6. In this exercise, you will test exception handling with the `SAVED EXCEPTIONS` clause.

   Run the `lab_07_06a.sql` file to create a test table:

```
CREATE TABLE card_table
(accepted_cards VARCHAR2(50) NOT NULL);
```

Open the `lab_07_06b.sql` file and run the contents:

```
DECLARE
  type typ_cards is table of VARCHAR2(50);
  v_cards typ_cards := typ_cards
  ( 'Citigroup Visa', 'Nationscard MasterCard',
    'Federal American Express', 'Citizens Visa',
    'International Discoverer', 'United Diners Club' );
BEGIN
  v_cards.Delete(3);
  v_cards.DELETE(6);
  FORALL j IN v_cards.first..v_cards.last
    SAVE EXCEPTIONS
    EXECUTE IMMEDIATE
   'insert into card_table (accepted_cards) values ( :the_card)'
    USING v_cards(j);
/
```

Note the output: This returns an "Error in Array DML (at line 11)," which is not very informational. The cause of this error is: one or more rows failed in the DML.

Oracle University and En-Sof Informatica E Treinamento Ltda use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

6. (continued)

Open the `lab_07_06c.sql` file and run the contents:

```
DECLARE
  type typ_cards is table of VARCHAR2(50);
  v_cards typ_cards := typ_cards
  ( 'Citigroup Visa', 'Nationscard MasterCard',
    'Federal American Express', 'Citizens Visa',
    'International Discoverer', 'United Diners Club' );
  bulk_errors EXCEPTION;
  PRAGMA exception_init (bulk_errors, -24381 );
BEGIN
  v_cards.Delete(3);
  v_cards.DELETE(6);
  FORALL j IN v_cards.first..v_cards.last
    SAVE EXCEPTIONS
    EXECUTE IMMEDIATE
   'insert into card_table (accepted_cards) values ( :the_card)'
    USING v_cards(j);
 EXCEPTION
  WHEN  bulk_errors THEN
    FOR j IN 1..sql%bulk_exceptions.count
  LOOP
    Dbms_Output.Put_Line (
      TO_CHAR( sql%bulk_exceptions(j).error_index ) || ':
      ' || SQLERRM(-sql%bulk_exceptions(j).error_code) );
  END LOOP;
END;
/
```

Note the output:

```
ORA-22160: element at index [] does not exist

PL/SQL procedure successfully completed.
```

Why is the output different?

The PL/SQL block raises the exception `22160` when it encounters an array element that was deleted. The exception is handled and the block completes successfully.

Oracle University and En-Sof Informatica E Treinamento Ltda use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 8: Solutions

In this exercise, you will profile the CREDIT_CARD_PKG package created in an earlier lesson.

1. Run the lab_08_01.sql script to create the CREDIT_CARD_PKG package.

```
@$HOME/labs/lab_08_01.sql
```

2. Run the proftab.sql script to create the profile tables under your schema.

```
@$HOME/labs/proftab.sql
```

3. Create a MY_PROFILER procedure to:

   Start the profiler

   Run the application

```
EXECUTE credit_card_pkg.update_card_info -
        (130, 'AM EX', 121212121212)
```

   Flush the profiler data

   Stop the profiler

```
CREATE OR REPLACE PROCEDURE my_profiler
(p_comment1 IN VARCHAR2, p_comment2 IN VARCHAR2)
IS
  v_return_code    NUMBER;
BEGIN
--start the profiler
  v_return_code:=DBMS_PROFILER.START_PROFILER
    (p_comment1, p_comment2);
  dbms_output.put_line
    ('Result from START: '||v_return_code);

-- now run a program...
  credit_card_pkg.update_card_info (130, 'AM EX', 121212121212);
--flush the collected data to the dictionary tables
  v_return_code := DBMS_PROFILER.FLUSH_DATA;
  dbms_output.put_line
    ('Result from FLUSH: '||v_return_code);
--stop profiling
  v_return_code := DBMS_PROFILER.STOP_PROFILER;
  dbms_output.put_line
    ('Result from STOP: '||v_return_code);
END;
/
```

Oracle University and En-Sof Informatica E Treinamento Ltda use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

4. Execute the MY_PROFILER procedure.

```
SET SERVEROUTPUT ON

EXECUTE my_profiler('Benchmark Run.' , 'This is the first run.')
```

5. Analyze the results of profiling in the PLSQL_PROFILER tables.

```
SELECT runid, run_owner, run_date, run_comment,
       run_comment1, run_total_time
FROM   plsql_profiler_runs;

SELECT runid, unit_number, unit_type,
       unit_owner, unit_name
FROM   plsql_profiler_units inner
JOIN   plsql_profiler_runs
USING  ( runid );

SELECT line#, total_occur, total_time,
       min_time, max_time
FROM   plsql_profiler_data
WHERE  runid = 1 AND unit_number = 2;
```

In this exercise, you will trace the CREDIT_CARD_PKG package.

6. Enable the CREDIT_CARD_PKG for tracing by using the ALTER statement with the COMPILE DEBUG option.

```
ALTER PACKAGE credit_card_pkg COMPILE DEBUG BODY;
```

7. Start the trace session and trace all calls.

```
EXECUTE DBMS_TRACE.SET_PLSQL_TRACE(DBMS_TRACE.trace_all_calls)
```

8. Run the credit_card_pkg.update_card_info procedure with the following data:

```
EXECUTE credit_card_pkg.update_card_info -
        (135, 'DC', 987654321)
```

Oracle University and En-Sof Informatica E Treinamento Ltda use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

9. Disable tracing.

```
EXECUTE DBMS_TRACE.CLEAR_PLSQL_TRACE
```

10. Examine the trace information by querying the trace tables.

```
COLUMN event_comment   format a28
COLUMN event_proc_name format a18
COLUMN proc_name       format a17

SELECT proc_name, proc_line,
         event_proc_name, event_comment
FROM sys.plsql_trace_events
WHERE event_unit = 'CREDIT_CARD_PKG';
```

Oracle University and En-Sof Informatica E Treinamento Ltda use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and En-Sof Informatica E Treinamento Ltda use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED