B

LAB SOLUTIONS

EXERCISE 1-1 SOLUTIONS

Evaluate each of the following declarations. Determine which of the following are *not* legal and explain why.

1
```
DECLARE
my_var   NUMBER(7,2);
```

2
```
DECLARE
x, y, z  CHAR(10);
```

Only one identifier per declaration allowed.

3
```
DECLARE
birthdate DATE NOT NULL;
```

4
```
DECLARE
in_stock  BOOLEAN := 1;
```

Must have a value assigned.

5
```
DECLARE
emp_rec  emp_rec_type;
```

emp_rec_type must be declared.

6
```
DECLARE
TYPE  dname_tab_type IS TABLE OF CHAR(20)
     INDEX BY BINARY_INTEGER;
dname_tab  dname_tab_type;
```

```
DECLARE
   weight      NUMBER := 600;
   location    CHAR(10) :='Europe';

BEGIN
   SUB-BLOCK 1
   DECLARE
      weight      NUMBER := 1;
      new_loc     CHAR(20);

   BEGIN
      weight := weight + 1;          (1)
      new_loc:= 'Western ' || location;   (2)
   END;

   SUB-BLOCK 2
   DECLARE
      new_wt      NUMBER;
      continent   CHAR(10);

   BEGIN
      continent := location;         (3)
      new_wt := weight + 10;         (4)
   END;

   weight := weight - 50;            (5)
   location := new_loc || ' and Asia';   (6)

END;
```

Exercise 1-1 Solutions—cont'd

Evaluate the PL / SQL block on the opposite page. Apply the scoping guidelines presented in this module to determine the values indicated:

1. **2**
2. **Western Europe**
3. **Europe**
4. **610**
5. **550**
6. **Illegal! (new_loc not visible outside of block 1)**

LAB 3-1 SOLUTIONS

Use the provided table descriptions below to write a PL / SQL block to satisfy the scenario below. Test your block by executing it within SQL*Plus.

Note: A VI and EDT editor reference handout appear on pages 3-12 and 3-13.

1 Get an order number from operator input and find the associated PRODUCT_ID. Search the INVENTORY table. If the product is IN STOCK, place a date into the order's associated ARRIVAL_DATE which is 7 days from today's date. [Use `SYSDATE + 7`]

2 If the product is BACK ORDERED, place a date into the order's associated ARRIVAL_DATE which is one month from today's date. [Use `ADD_MONTHS(SYSDATE, 1)`].

3 If the product is a SPECIAL ORDER, place a date two months from today's date into the order's associated ARRIVAL_DATE [use `ADD_MONTHS(SYSDATE,2)`], and fulfill the special order by inserting a row into the SPECIAL_ORDERS table. Use the ORDER_NO from the CUST_ORDERS table. Use the standard order quantity (STD_ORDER_QTY) in the order.

INVENTORY TABLE

[NUMBER(6)] PRODUCT_ID	[CHAR(30)] PRODUCT_DESCRIPTION	[CHAR(20)] PRODUCT_STATUS	[NUMBER(3)] STD_ORDER_QTY
1	JACKET STYLE #1	IN STOCK	100
2	JACKET STYLE #2	BACK ORDERED	200
3	JACKET STYLE #3	SPECIAL ORDER	300

CUST_ORDERS TABLE

[NUMBER(12)] ORDER_NO	[NUMBER(6)] PRODUCT_ID	[DATE] ARRIVAL_DATE
1	1	
2	1	
3	2	
4	1	
5	2	
6	3	

Lab 3-1 Solutions—cont'd

SPECIAL_ORDERS TABLE

[NUMBER(12)]	[NUMBER(6)]	[NUMBER(3)]
ORDER_NO	PRODUCT_ID	ORDER_QTY
--------	----------	---------

```
DECLARE
  inventory_rec    inventory%rowtype;

BEGIN
  SELECT * INTO inventory_rec FROM inventory WHERE product_id =
     (SELECT product_id FROM cust_orders
        WHERE order_no = &&orderno);

  IF inventory_rec.product_status = 'IN STOCK' THEN
     UPDATE cust_orders SET arrival_date = SYSDATE+7
        WHERE order_no = &&orderno;

  ELSIF inventory_rec.product_status = 'BACK ORDERED' THEN
     UPDATE cust_orders SET arrival_date = ADD_MONTHS(SYSDATE,1)
        WHERE order_no = &&orderno;

  ELSIF inventory_rec.product_status = 'SPECIAL ORDER' THEN
     UPDATE cust_orders SET arrival_date = ADD_MONTHS(SYSDATE,2)
        WHERE order_no = &&orderno;
     INSERT INTO special_orders VALUES
        (&&orderno,inventory_rec.product_id,
           inventory_rec.std_order_qty);
  END IF;
  COMMIT;
END;
.
/
```

LAB 3-2 SOLUTIONS

Use the provided table description below to write a PL / SQL block to satisfy the scenario below. Test your block by executing it within SQL*Plus.

1 Write a PL/SQL procedure that has an outer loop and an inner loop. The outer loop iterates 5 times, and the inner loop iterates 4 times.

2 In the outer loop, insert a row into the TEMP table (depicted below) that has the loop's index in COL1, NULL in COL2, and the message *In outer loop* in MESSAGE.

3 In the inner loop, insert a row into the TEMP table that contains the outer loop's index in COL1, the inner loop's index in COL2, and the message *In inner loop* in MESSAGE.

```
                              TEMP TABLE

[NUMBER(9,4)] [NUMBER(9,4)] [CHAR(55)]
         COL1          COL2  MESSAGE
     --------      --------  ---------
```

```
BEGIN
  <<outer>>
  FOR i IN 1..5 loop
     INSERT INTO temp VALUES (i, NULL, 'In outer loop');
     <<inner>>
     FOR j IN 1..4 loop
        INSERT INTO temp VALUES(i, j, 'In inner loop');
     END LOOP inner;
  END LOOP outer;
END;
.
/
```

LAB 4-1 SOLUTIONS

Write a PL / SQL block to satisfy the scenario below. Test your block by executing it within SQL*Plus.

1 Get a number *n* from the operator, using an & variable.

2 Using a simple loop, get the name and salary of the people with the top *n* salaries in the company. All names and salaries may be found in the EMP table (see below).

EMP

[CHAR(10)]	[NUMBER(7,2)]
ENAME	SAL
SMITH	800
ALLEN	1600
WARD	1250
JONES	2975
MARTIN	1250
BLAKE	2850
CLARK	2450
SCOTT	3000
KING	5000
TURNER	1500
ADAMS	1100
JAMES	950
FORD	3000
MILLER	1300

3 Store them in a table called TOP_SALS, with columns NAME CHAR(20) and SALARY NUMBER(7,2).

4 Repeat the procedure using a cursor FOR loop.

Lab 4-1 Solutions—cont'd

```
DECLARE
  --Simple Loop
  CURSOR emp_cur IS SELECT ename, sal
     FROM emp
     ORDER BY sal DESC;
  emp_rec emp_cur%ROWTYPE;
BEGIN
  OPEN emp_cur;
  LOOP
     FETCH emp_cur INTO emp_rec;
     EXIT WHEN (emp_cur%NOTFOUND);
     INSERT INTO top_sals VALUES(emp_rec.ename,emp_rec.sal);
     EXIT WHEN (emp_cur%ROWCOUNT = &&enter_how_many);
  END LOOP;
  CLOSE emp_cur;
  COMMIT;
END;
.
/
```

```
DECLARE
  --Cursor FOR loop
  CURSOR emp_cur IS SELECT ename,sal FROM emp
  ORDER BY sal DESC;

BEGIN
  FOR emp_rec IN emp_cur LOOP
     EXIT WHEN (emp_cur%ROWCOUNT > &&count);
     INSERT INTO top_sals values(emp_rec.ename,
     emp_rec.sal);
  END LOOP;
  COMMIT;
END;
.
/
```

LAB 5-1 SOLUTIONS

Write a PL / SQL block based upon the tables depicted below and the scenario below. Test your block by executing it in SQL*Plus.

1 The ACTION table defines a set of actions to be taken on the ACCOUNTS table. The actions (stored in the oper_type column) may either be 'I' to insert a new account, or 'U' to update an existing account. For example, if a row in ACTION reads 6, 'I', 2099, that means to create account 6 with $2099 as an initial balance. If there was a 'U' instead of an 'I' in the same row, it would mean update account 6, and set the new balance to $2099.

2 Write a PL/SQL block that goes through each row of ACTION (Hint: You'll need a cursor here) and carries out the actions specified.

3 If you attempt to insert a row that already exists, perform an update instead. The DUP_VAL_ON_INDEX internal exception will be raised, so your block should include a local handler to perform the update. If you perform an update on a row that doesn't exist, perform an insert instead. How can you tell if no rows were updated?

ACCOUNTS TABLE

UNIQUE INDEX

[NUMBER(4)] ACCOUNT_ID	[NUMBER(11,2)] BAL
1	1000
2	2000
3	1500
4	6500
5	500

ACTION TABLE

[NUMBER(4)] ACCOUNT_ID	[CHAR(1)] OPER_TYPE	[NUMBER(11,2)] NEW_VALUE
3	U	599
6	I	2099
7	U	1599
1	I	399

Lab 5-1 Solutions—cont'd

```
DECLARE
  CURSOR action_cur IS
     SELECT account_id, oper_type, new_value FROM action;
BEGIN
  FOR acct IN action_cur LOOP
           -- process each row, one at a time
     /*-----------------------------------------------------*
      * Process an UPDATE. If the account to be updated     *
      *  doesn't exists, create a new account.
  *
      *-----------------------------------------------------*/
     IF acct.oper_type = 'U' THEN
        UPDATE accounts SET bal = acct.new_value
           WHERE account_id = acct.account_id;
        IF SQL%NOTFOUND THEN
           INSERT INTO accounts
              VALUES (acct.account_id, acct.new_value);
        END IF;
     /*-----------------------------------------------------*
      * Process an INSERT. If the account already exists,   *
      *  do an update of the account instead.               *
      *-----------------------------------------------------*/
     ELSIF acct.oper_type = 'I' THEN
        BEGIN
           INSERT INTO accounts
              VALUES (acct.account_id, acct.new_value);

        EXCEPTION
           WHEN DUP_VAL_ON_INDEX THEN -- account already exists
              UPDATE accounts SET bal = acct.new_value
                   WHERE account_id = acct.account_id;
        END;
     END IF;
  END LOOP;
  COMMIT;
END;
.
/
```

Note: *An alternate solution is on the next page.*

Lab 5-1 Solutions—cont'd

```
DECLARE
  dummy accounts.account_id%TYPE;
  CURSOR action_cur IS
     SELECT account_id, oper_type, new_value FROM action;
BEGIN
  FOR acct IN action_cur LOOP
           -- process each row, one at a time
     /*-----------------------------------------------------*
      * Process an UPDATE. If the account to be updated     *
      *  doesn't exists, create a new account.
  *
      *-----------------------------------------------------*/
     IF acct.oper_type = 'U' THEN
        UPDATE accounts SET bal = acct.new_value
           WHERE account_id = acct.account_id;
        IF SQL%NOTFOUND THEN
           INSERT INTO accounts
              VALUES (acct.account_id, acct.new_value);
        END IF;
     /*-----------------------------------------------------*
      * Process an INSERT. If the account already exists,   *
      *  do an update of the account instead.               *
      *-----------------------------------------------------*/
     ELSIF acct.oper_type = 'I' THEN
        BEGIN
           SELECT account_id INTO dummy
           FROM accounts
           WHERE account_id = acct.account_id;
           UPDATE accounts SET bal = acct.new_value
              WHERE account_id = acct.account_id;
        EXCEPTION
           WHEN NO_DATA_FOUND THEN -- account does not exist
              INSERT INTO accounts
                   VALUES (acct.account_id, acct.new_value);
        END;
     END IF;
  END LOOP;
  COMMIT;
END;
.
/
```

Optional Lab

Write a PL / SQL block to satisfy the scenario below. Test your block by executing it in SQL*Plus.

4 Prompt the user for a department number, and then, using a cursor, get the salary, commission, and name of each employee in the department. This information is inserted into the COL1, COL2, and MESSAGE columns of the TEMP table (depicted below).

5 However, if an employee in that department has a NULL commission,raise a user-defined exception. In the exception handler, first ROLLBACK any previous INSERTs. Then CLOSE and re-OPEN the cursor, this time inserting only the salary and name of each employee into the COL1 and MESSAGE columns of the TEMP table.

```
                    TEMP TABLE

[NUMBER(9,4)]   [NUMBER(9,4)]      [CHAR(55)]
        COL1            COL2        MESSAGE
    --------        --------        ---------
```

Lab 5-1 Solutions—cont'd

```
DECLARE
  CURSOR emp_cur IS
     SELECT sal, comm, ename from emp WHERE deptno = &enter_deptno;
  null_commission EXCEPTION;

BEGIN
  FOR emprec IN emp_cur LOOP
     INSERT INTO temp (col1, col2, message)
        VALUES (emprec.sal, emprec.comm, emprec.ename);
     IF emprec.comm IS NULL THEN
     RAISE null_commission;
     END IF;
  END LOOP;

EXCEPTION
  WHEN null_commission THEN
     ROLLBACK;
     FOR emprec IN emp_cur LOOP
        INSERT INTO temp (col1, message)
           VALUES (emprec.sal, emprec.ename);
     END LOOP;
     COMMIT;
END;
```

Lab 6-1 Solutions—cont'd

```
BEGIN
FOR ACCT IN ACTION_CUR LOOP
  -- PROCESS EACH ROW, ONE AT A TIME
     IF ACCT.OPER_TYPE = 'U' THEN
        UPDATE_ACCT (ACCT.NEW_VALUE, ACCT.ACCOUNT_ID);
     ELSIF ACCT.OPER_TYPE = 'I' THEN
        INSERT_ACCT (ACCT.NEW_VALUE, ACCT.ACCOUNT_ID);
     ELSIF ACCT.OPER_TYPE - 'D' THEN
        DELETE_ACCT (ACCT.NEW_VALUE, ACCT.ACCOUNT_ID):
     END IF;
END LOOP;
END;
.
/
```

LAB 6-1 SOLUTIONS

Write a PL / SQL block based upon the tables and scenario in lab 3-1 given the additional information below. Test your block by executing it in SQL*Plus.

1 The ACTION table defines a set of actions to be taken on the ACCOUNTS table. The actions (stored in the OPER_TYPE column) may either be 'I' to insert a new account, 'U' to update an existing account, or 'D' to delete an existing account. On an insert, if the account already exists, an update is done instead. On an update, if the account does not exist, it is created by an insert. On a delete, if the row does not exist, no action is taken.

2 Write a PL/SQL block that goes through each row of ACTION and carries out the actions specified. Incorporate procedures in the declaration section of the block for each of the different actions (insert, update, and delete) and call the procedures from the executable section of the block (Hint: You'll need to pass parameters to the procedures).

UNIQUE INDEX
ACCOUNTS TABLE

[NUMBER(4)]	[NUMBER(11,2)]
ACCOUNT_ID	BAL
1	1000
2	2000
3	1500
4	6500
5	500

ACTION TABLE

[NUMBER(4)]	[CHAR(1)]	[NUMBER(11,2)]
ACCOUNT_ID	OPER_TYPE	NEW_VALUE
3	U	599
6	I	2099
5	D	
7	U	1599
1	I	399
9	D	

```
DECLARE
  CURSOR ACTION_CUR IS
     SELECT ACCOUNT_ID, OPER_TYPE, NEW_VALUE FROM ACTION;
  PROCEDURE UPDATE_ACCT (NEW_VAL IN NUMBER, ID IN NUMBER) IS
  /*-------------------------------------------------------------*
* PROCESS AN UPDATE.  IF THE ACCOUNT TO BE UPDATED              *
* DOESN'T EXIST, CREATE A NEW ACCOUNT.                          *
  *-------------------------------------------------------------*/
  BEGIN
     UPDATE ACCOUNTS SET BAL = NEW_VAL
        WHERE ACCOUNT_ID = ID;
     IF SQL%NOTFOUND THEN
        INSERT INTO ACCOUNTS
           VALUES (ID, NEW_VAL);
     END IF;
  END UPDATE_ACCT;

  PROCEDURE INSERT_ACCT (NEW_VAL IN NUMBER, ID IN NUMBER) IS
/*-------------------------------------------------------------*
  * PROCESS AN INSERT.  IF THE ACCOUNT ALREADY EXISTS,          *
  * DO AN UPDATE OF THE ACCOUNT INSTEAD.                        *
  *-------------------------------------------------------------*/
  BEGIN
     INSERT INTO ACCOUNTS
        VALUES (ID, NEW_VAL);
  EXCEPTION
     WHEN DUP_VAL_ON_INDEX THEN
        UPDATE ACCOUNTS SET BAL = NEW_VAL
           WHERE ACCOUNT_ID = ID;
  END INSERT_ACCT;

  PROCEDURE DELETE_ACCT (NEW_VAL IN NUMBER, ID IN NUMBER) IS
/*-------------------------------------------------------------*
  * PROCESS A DELETE.                                           *
  *-------------------------------------------------------------*/
  BEGIN
     DELETE FROM ACCOUNTS
        WHERE ACCOUNT_ID = ID;
  END DELETE_ACCT;
```

solution continued on next page