**Additional Practices 1 and 2**

1. Evaluate each of the following declarations. Determine which of them are not legal and explain why
   1. DECLARE

name, dept VARCHAR2(14);

* 1. DECLARE

test NUMBER(5);

* 1. DECLARE

MAXSALARY NUMBER(7,2) := 5000;

* 1. DECLARE

JOINDATE BOOLEAN := SYSDATE;

1. In each of the following assignments, determine the data type of the resulting expression
   1. email := firstname || to\_char(empno);
   2. confirm := to\_date(’20-JAN-1999’, ‘DD-MON-YYYY’);
   3. sal := (1000\*12) + 500;
   4. test := FALSE;
   5. temp := temp1 < (temp2/3);
   6. var := sysdate;

**Additional Practice 3**

1. DECLARE

v\_custid NUMBER(4) := 1600;

v\_custname VARCHAR2(300) := ‘Women Sports Club’;

v\_new\_custid NUMBER (3) := 500;

BEGIN  
 DECLARE  
 v\_custid NUMBER(4) := 0;  
 v\_custname VARCHAR2(300) := ‘Shape up Sports Club’;  
 v\_new\_custid NUMBER(3) := 300;  
 v\_new\_custname VARCHAR2(300) := ‘Jansports Club’;  
 BEGIN  
 v\_custid := v\_new\_custid;  
 v\_custname := v\_custname || ‘ ‘ || v\_new\_custname;

1  
 END;  
 v\_custid := (v\_custid \* 12) / 10;

2

END;  
/

Evaluate the PL/SQL block given above and determine the data type and value of each of the following variables according to the rules of scoping::

1. The value of v\_custid at position 1:
2. The value of v\_custname at position 1:
3. The value of v\_new\_custid at position 2:
4. The value of v\_new\_custname at position 1:
5. The value of v\_custid at position 2:
6. The value of v\_custname at position 2:

**Additional Practice 4 and 5**

1. Write a PL/SQL block to accept a year and check whether it is a leap year. For example, if the year entered is 1990, the output should be “1990 is not a leap year.”  
   Hint: the year should be exactly divisible by 4 but not divisible by 100, or it should be divisible by 400.

Test your solution with the following years:

|  |  |
| --- | --- |
| 1990 | Not a leap year |
| 2000 | Leap year |
| 1996 | Leap year |
| 1886 | Not a leap year |
| 1992 | Leap year |
| 1824 | Leap year |

**anonymous block completed  
1990 is not a leap year**

1. a. For the exercise below, you must create a temporary table to store the results. Create a table named TEMP with the following three columns:

|  |  |  |  |
| --- | --- | --- | --- |
| Column Name | NUM\_STORE | CHAR\_STORE | DATE\_STORE |
| Key Type |  |  |  |
| Nulls/Unique |  |  |  |
| FK Table |  |  |  |
| FK Column |  |  |  |
| Data Type | Number | Varchar2 | Date |
| Length | 7,2 | 35 |  |

b. Write a PL/SQL block that contains two variables, V\_MESSAGE and V\_DATE\_WRITTEN. Declare V\_MESSAGE as VARCHAR2 data type with a length of 35 and V\_DATE\_WRITTEN as DATE data type. Assign the following values to the variables:

Variable Contents

V\_MESSAGE This is my first PL/SQL program

V\_DATE\_WRITTEN Current date

Store the values in appropriate columns of the TEMP table. Verify your results by querying the TEMP table.

**Additional Practice 6 and 7**

1. a. Store a department number in a substitution variable

b. Write a PL/SQL block to print the number of people working in that department

**anonymous block completed**

**6 employee(s) work for the department number 30**

1. Write a PL/SQL block to declare a variable called v\_sal to store the salary of an employee. In the executable part of the program, do the following:
   1. Store an employee name in a substitution variable
   2. Store his or her salary in the v\_sal variable
   3. If the salary is less than 3,000, give the employee a raise of 5000 and display the message ‘<Employee name>’s salary updated” in the window
   4. If the salary is more than 3,000, print the employee’s salary in the format , “<Employee name> earns …………..”
   5. Test the PL/SQL block for the following last names:

|  |  |
| --- | --- |
| LAST\_NAME | SALARY |
| Pataballa | 4800 |
| Greenberg | 12000 |
| Ernst | 6000 |

1. Write a PL/SQL block to store the salary of an employee(choose the name at runtime) in a variable. In the executable part of the program, do the following:

* Calculate the annual salary as salary \* 12
* Calculate the bonus as indicated below:

|  |  |
| --- | --- |
| Annual Salary | Bonus |
| >=150,000 | 2,000 |
| 149,999 – 100,000 | 1,000 |
| <=99,999 | 500 |

Display the amount of the bonus in the window in the following format:  
 “The bonus is $..........................”

* Test the PL/SQL for the following test cases:

|  |
| --- |
| Employee |
| John Chen |
| Alexander Khoo |
| Janette King |

1. a. Create a temporary table called emp.

DROP TABLE emp;  
CREATE TABLE emp AS SELECT \* FROM EMPLOYEES;

Write a PL/SQL block to store an employee number, the new department number, and the percentage increase in the salary in substitution variables

b. Update the department ID of the employee with the new department number, and update the salary with the new salary. Use the emp table for the updates. After the update is complete, display the message “Update complete” in the window. If no matching records are found, display “No data found”. Test the PL/SQL block for the following test cases:

|  |  |  |  |
| --- | --- | --- | --- |
| EMPLOYEE\_ID | NEW\_DEPARTMENT\_ID | % INCREASE | MESSAGE |
| 100 | 20 | 2 | Update complete |
| 10 | 30 | 5 | No data found |
| 126 | 40 | 3 | Update complete |

c. Create a PL/SQL block to display the employee name, salary and hire date “in the format” shown in the sample output of practice 10. Do this for the employees with ID 105 – 110 – 115 – 120 – 125 but only if the salary is greater than 5,000 and the hire date is later than 01-02-1988.

**Additional Practice 10 and 11**

1. Create a PL/SQL block to declare a cursor EMP\_CUR to select the employee name, salary, and hire date from the employees table. Process each row from the cursor, and if the salary is greater than 15,000 and the hire date is later than 01-FEB-1988, display the employee name, salary and hire date in the window in the format shown in the sample output below:

**anonymous block completed  
Kochhar earns 17000 and joined the organization on 21-SEP-89  
De Haan earns 17000 and joined the organization on 13-JAN-93**

1. Create a PL/SQL block to retrieve the last name and department ID for each employee from the EMPLOYEES table for those employees whose EMPLOYEE\_ID is less than 114. With the values retrieved from the employees table, populate two PL/SQL tables, one to store the records of the employee last names and the other to store the records of their department IDs. Using a loop, retrieve the employee name information and the department ID information from the PL/SQL tables and display it in the window, using DBMS\_OUTPUT.PUT\_LINE. Display these details for the first 15 employees in the PL/SQL tables.

**anonymous block completed  
Employee Name: King Department\_id: 90  
Employee Name: Kochhar Department\_id:90  
Employee Name: De Haan Department\_id:90  
Employee Name: Hunold Department\_id:60  
Employee Name: Ernst Department\_id:60  
Employee Name: Austin Department\_id:60  
Employee Name: Pataballa Department\_id:60  
Employee Name: Lorentz Department\_id:60  
Employee Name: Greenberg Department\_id:100  
Employee Name: Faviet Department\_id:100  
Employee Name: Chen Department\_id:100  
Employee Name: Sciarra Department\_id:100  
Employee Name: Urman Department\_id:100  
Employee Name: Popp Department\_id:100  
Employee Name: Raphaely Department\_id:30**

1. a. Create a PL/SQL block that declares a cursor called DATE\_CUR. Pass a parameter of the DATE data type to the cursor and print the details of all the employees who have joined after that date.

DEFINE B\_HIREDATE = 08-MAR-00

b. Test the PL/SQL block for the following hire dates: 08-MAR-00, 25-JUN-97, 28-SEP-98, 07-FEB-99.

**anonymous block completed  
166 Ande 24-MAR-00  
167 Banda 21-APR-00  
173 Kumar 21-APR-00**

1. Re-create the emp table:

DROP TABLE emp;  
CREATE TABLE emp AS SELECT \* FROM EMPLOYEES;

Create a PL/SQL block to promote clerks who earn more than 3,000 to the job title SR CLERK and increase their salaries by 10%.Use the EMP table for this practice. Verify the result by querying the emp table.

Hint: Use a cursor with the FOR UPDATE and CURRENT OF syntaxes.

1. a. For the exercise below, you will require a table to store the results. You can create the analysis table by the following code:

CREATE TABLE analysis

(ename varchar2(20),

years number(2),

sal number(8,2));

b. Create a PL/SQL block to populate the analysis table with the information from the employees table. Use a substitution variable to store an employee’s last name.

c. Query the employees table to find out whether the number of years that the employee has been with the organization is greater than five; and if the salary is less than 3,500, raise an exception. Handle the exception with an appropriate exception handler that inserts the following values into the analysis table:  
employee last name, number of years of service, and the current salary. Otherwise display ‘Not due for a raise’ in the window. Verify the results by querying the analysis table. Use the following test cases tot test the PL/SQL block:

|  |  |
| --- | --- |
| LAST\_NAME | MESSAGE |
| Austin | Not due for a raise |
| Nayer | Due for a raise |
| Fripp | Not due for a raise |
| Khoo | Due for a raise |

Additional Practices

1. In this exercise, create a program to add a new job into the JOBS table
   1. Create a stored procedure called NEW\_JOB to enter a new order into the JOBS table. The procedure should accept three parameters. The first and second parameters supply a job ID and a job title. The third parameter supplies the minimum salary. Use the maximum salary for the new job as twice the minimum salary supplied for the job ID.
   2. Invoke the procedure to add a new job with job ID ‘SY\_ANAL’, job\_title ‘System Analyst’ and minimum salary 6000.
   3. Verify that a row was added, and note the new job ID for use in the next exercise. Commit the changes
2. In this exercise, create a program to add a new row to the JOB\_HISTORY table for an existing employee.
   1. Create a stored procedure called ADD\_JOB\_HIST to add a new row into the JOB\_HISTORY table for an employee who is changing his job for the new job ID (‘SY\_ANAL’) that you created in the previous exercise.

The procedure should provide two parameters: one for the employee ID who is changing the job, and the second for the new job ID. Read the employee ID from the EMPLOYEES table and insert it into the JOB\_HISTORY table. Make the hire date of this employee as the start date and today’s date as the end date for this row in the JOB\_HISTORY table.

Change the hire date of this employee in the EMPLOYEES table to today’s date. Update the job ID of this employee to the job ID passed as parameter (use the ‘SY\_ANAL’ job ID) and salary to equal to the minimum salary for that job ID plus 500.

**Note**: include exception handling to handle an attempt to insert a nonexistent employee.

* 1. Disable all triggers on the EMPLOYEES, JOBS, and JOB\_HISTORY tables before invoking the ADD\_JOB\_HIST procedure
  2. Execute the procedure with employee ID 106 and job ID ‘SY\_ANAL’ as parameters.
  3. Query the JOB\_HISTORY and EMPLOYEES tables to view your changes for employee 106, and then commit the changes.
  4. Reenable the triggers on the EMPLOYEES, JOBS and JOB\_HISTORY tables

1. In this exercise, create a program to update the minimum and maximum salaries for a job in the JOBS table
   1. Create a stored procedure called UPD\_JOBSAL to update the minimum and maximum salaries for a specific job ID in the JOBS table. The procedure should provide three parameters: the job ID, a new minimum salary, and a new maximum salary. Add exception handling to account for an invalid job ID in the JOBS table. Raise an exception if the maximum salary supplied is less than the minimum salary. Provide a message that will be displayed if the row in the JOBS table is locked.  
      Hint: The resource locked/busy error number is -54
   2. Execute the UPD\_JOBSAL procedure by using a job ID of ‘SY\_ANAL’, a minimum salary of 7000, and a maximum salary of 14000.  
      Note: This should generate an exception message
   3. Disable triggers on the EMPLOYEES and JOBS tables
   4. Execute the UPD\_JOBSAL procedure using a job ID of ‘SY\_ANAL’, a minimum salary of 7000, and a maximum salary of 14000
   5. Query the JOBS table to view your changes, and then commit the changes
   6. Enable the triggers on the EMPLOYEES and JOBS tables
2. In this exercise, create a procedure to monitor whether employees have exceeded their average salaries for their job type
   1. Disable the SECURE\_EMPLOYEES trigger
   2. In the EMPLOYEES table, add an EXCEED\_AVGSAL column for storing up to three characters and a default value of NO. Us a check constraint to allow the values YES of NO.
   3. Write a stored procedure called CHECK\_AVGSAL that checks whether each employee’s salary exceeds the average salary for the JOB\_ID. The average salary for a job is calculated from information in the JOBS table. If the employee’s salary exceeds the average for his of her job, then update his or her EXCEED\_AVGSAL column in the EMPLOYEES table to a value of YES; otherwise, set the value to NO. Us a cursor to select the employee’s rows using the FOR UPDATE option in the query. Add exception handling to account for a record being locked.  
      Hint: the resource locked/busy error number is -54. Write and use a local function called GET\_JOB\_AVGSAL to determine the average salary for a job ID specified as a parameter.
   4. Execute the CHECK\_AVGSAL procedure. Then, to view the results of your modifications, write a query to display the employee’s ID, job, the average salary for the job, the employee’s salary, and the exceed\_avgsal indicator column for employees whose salaries exceed the average for their job, and finally commit the changes.
3. Create a subprogram to retrieve the number of years of service for a specific employee
   1. Create a stored procedure called GET\_YEARS\_SERVICE to retrieve the total number of years of service for a specific employee. The function should accept the employee ID as a parameter and return the number of years of service. Add error handling to account for an invalid employee ID.
   2. Invoke the GET\_YEARS\_SERVICE function in al call to DBMS\_OUTPUT.PUT\_LINE for an employee with ID 999
   3. Display the number of years of service for employee 106 with DBMS\_OUTPUT.PUT\_LINE invoking the GET\_YEARS\_SERVICE function
   4. Query the JOB\_HISTORY and EMPLOYEES table for the specified employee to verify that the modifications are accurate.  
      Note: The values represented in the results on this page may differ from those you get when you run these queries.
4. In this exercise, create a program to retrieve the number of different jobs that an employee worked on during his or her service.
   1. Create a stored function called GET\_JOB\_COUNT to retrieve the total number of different jobs on which an employee worked

The function should accept the employee ID in a parameter, and return the number of different jobs that the employee worked on until now, including the present job. Add exception handling to account for an invalid employee ID.  
Hint: Use the distinct job IDs from the JOB\_HISTORY table, and exclude the current job ID, if it is one of the job IDs on which the employee has already worked. Write a UNION of two queries and count the rows retrieved into a PL/SQL table. Use a FETCH with BULK COLLECT INTO to obtain the unique jobs for the employee.

* 1. Invoke the function for an employee with ID 176.

1. Create a package calle EMPJOB\_PKG that contains your NEW\_JOB, ADD\_JOB\_HIST, UPD\_JOBSAL procedures, as well as your GET\_YEARS\_SERVICE and GET\_JOB\_COUNT functions.
   1. Create the package specifications with all the subprogram constructs as public. Move any subprogram local-defined types into the package specification
   2. Create the package body with the subprogram implementation; remember to remove, from the subprogram implementations, any types that you moved into the package specification
   3. Invoke your EMPJOB\_PKG.NEW\_JOB procedure to create a new job with the ID PR\_MAN, the job title Public Relations Manager, and the salary 6250
   4. Invoke your EMPJOB\_PKG.ADD\_JOB\_HIST procedure to modify the job of employee ID 110 to job ID PR\_MAN  
      Note: You need to disable the UPDATE\_JOB\_HISTORY trigger before you execute the ADD\_JOB\_HIST procedure, and re-enable the trigger after you have executed the procedure.
   5. Query the JOBS, JOB\_HISTORY, and EMPLOYEES tables to verify the results.
2. In this exercise, create a trigger to ensure that the minimum and maximum salaries of a job are never modified such that the salary of an existing employee with that job ID is outside the new range specified for the job.
   1. Create a trigger called CHECK\_SAL\_RANGE that is fired before every row that is updated in the MIN\_SALARY and MAX\_SALARY columns in the JOBS table. For any minimum or maximum salary value that is changed, check whether the salary of any existing employee with that job ID in the EMPLOYEES table falls within the new range of salaries specified for this job ID. Include exception handling to cover a salary range change that affects the record of any existing employee.