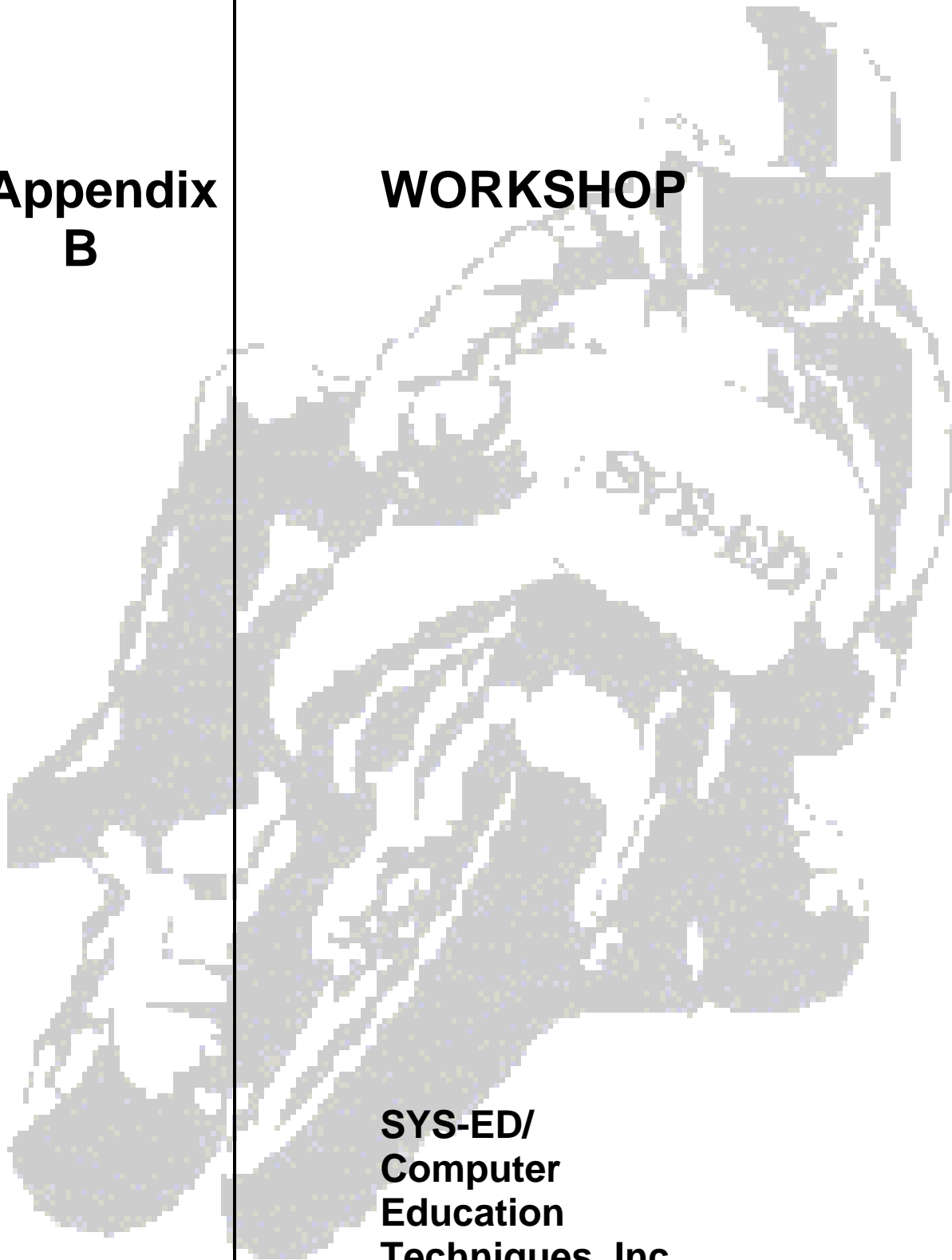


**Appendix
B**

WORKSHOP



**SYS-ED/
Computer
Education
Techniques, Inc.**

1 Variable Declaration and PL/SQL Block**Objectives:**

- Understand variable declaration.
- Understand PL/SQL block.
- Use PL/SQL.

Description:

1. Examine each of the following declarations. Determine which of are not valid and why?

a) DECLARE v_id NUMBER(4);

b) DECLARE v_a v_a, v_b VARCHAR2(14);

c) DECLARE v_purchasedate DATE DATE NOT NULL;

d) DECLARE v_taxable BOOLEAN 1;

2. Examine the following assignments:

- a) `v_days_left := v_due_date - SYSDATE;`
- b) `v_recipient := USER || ':' || TOCHAR(v_dept_no);`
- c) `v_sum := $25,300 + $120,000;`
- d) `v_flag := TRUE;`
- e) `v_m := v_n > (3 * v_p);`
- f) `v_value := NULL;`

Which of these assignments have valid statements.

What is the valid data type of the result for each assignment.

3. Create a block that declares two variables and then assign the value of these PL/SQL variables to iSQL*Plus host variables. Then print the results of the PL/SQL variables to the screen.

Execute the PL/SQL block.

Save the PL/SQL block in a file named plq4.sql, by clicking the Save Script button.

Save the script with a .sql extension.

V_CHAR Character (variable length)

V_NUM Number

Assign values to the following variables:

Variable	Value
V_CHAR	Computer Education Techniques, Inc.
V_NUM	500

2 Scoping and Nesting Rules and Evaluating PL/SQL Blocks

Objectives:

- Understand scoping and nesting rules.
- Understand PL/SQL blocks.

Description:

Given the following PL/SQL block:

```
DECLARE
    v_weight NUMBER(3):= 600;
    v_message VARCHAR2(255):='Product 10012';
BEGIN
    DECLARE
        v_weight          NUMBER(3) := 1;
        v_message         VARCHAR2(255) := 'Product 11001';
        v_newlocn         VARCHAR2(50) := 'Europe';
    BEGIN
        v_weight := v_weight+ 1;
        v_newlocn := 'Western ' || v_newlocn;
    END;
END;
```

Position A

```
v_weight := v_weight + 1;  
v_message := v_message || ' is in stock';  
vnewlocn := 'Western ' || v_newlocn;
```

Position B

END;

1. Evaluate the PL/SQL block above and determine the data type and value of each of the following variables according to the rules of scoping.
 - a) The value of V_WEIGHT at position A is _____.
 - b) The value of V_NEW_LOCN at position A is _____.
 - c) The value of V_WEIGHT at position B is _____.
 - d) The value of V_MESSAGE at position B is _____.
 - e) The value of V_NEW_LOCN at position B is _____.

Given the following PL/SQL block:

```
DECLARE
```

```
    v_customer    VARCHAR2(50) := 'Womansport';
```

```
    v_credit_rating VARCHAR2(50) := 'EXCELLENT';
```

```
BEGIN
```

```
DECLARE
```

```
    v_customer NUMBER(7) := 201;
```

```
    v_name      VARCHAR2(25) := 'Unisports';
```

```
BEGIN
```

```
    v_customer
```

```
    v_name
```

```
    v_credit_rating
```

```
    v_customer
```

```
    v_name
```

```
    v_credit_rating
```

```
END;
```

2. Use the embedded subblock within a block in order to declare two variables:

V_CUSTOMER and V_CREDIT_RATING, in the main block.

Declare two variables in the main block: V_CUSTOMER and V_NAME

Determine the values and data types for each of the following cases.

- a) The value of V_CUSTOMER in the subblock is _____.
- b) The value of V_NAME in the subblock is: _____.
- c) The value of V_CREDIT_RATING in the subblock is: _____.
- d) The value of V_CUSTOMER in the main block is: _____.
- e) The value of V_NAME in the main block is: _____.
- f) The value of V_CREDIT_RATING in the main block is: _____.

3 Code Block Creation

Objectives:

- Create a PL/SQL block to select data from a table.
- Create a PL/SQL block to insert data into a table.
- Create a PL/SQL block to update data in a table.
- Create a PL/SQL block to delete a record from a table.

Description:

1. Create a PL/SQL block that selects the maximum category id in the COURSES table and stores it in an /SQL*Plus variable. Print the results to the screen.
2. Modify the PL/SQL block created in exercise 1 to insert a new category into the COURSES table. Save the PL/SQL block in a file with .sql extension.
 - a) Rather than printing the category number retrieved from exercise 1, add 10 to it and use it as the category number for the new category.
 - b) Use the DEFINE command to provide the category name. Name the new category Internet. Pass the value to the PL/SQL block through an /SQL*Plus substitution variable.
 - c) Execute the PL/SQL block.
 - d) Display the new category that has been created.

3. Create a PL/SQL block that updates the category ID for the new category that you added in the previous practice.
 - a) Use an iSQL*Plus variable for the category ID number that you added in the previous practice.
 - b) Use the DEFINE command to provide the location ID. Name the new location id 400. Pass the value to the PL/SQL block through an iSQL*Plus substitution variable.
 - c) Test the PL/SQL block.

```
DEFINE p_catno = 280
```

```
DEFINE p_locno = 400
```

4. Create a PL/SQL block that deletes the department that was created in the previous exercise.

4 Conditional Actions - IF Statement Iterative Steps – Loop Structure

Objectives:

- Perform conditional actions using the IF statement.
- Perform iterative steps using the loop structure.

Description:

1. Create a table called MESSAGES with a field named MESSAGE_ID. Write a PL/SQL block to insert numbers into the MESSAGES table.
 - a) Insert the numbers 1 to 10, excluding 6 and 8 in MESSAGE_ID field.
 - b) Select from the MESSAGES table to verify that your PL/SQL block worked.
2. Create a PL/SQL block that computes the commission amount for a given employee based on the employee's salary.
 - a) Use the DEFINE command to provide the employee ID. Pass the value to the PL/SQL block through an iSQL*Plus substitution variable.

```
DEFINE p_empno = 100
```

- b) When the employee's salary is less than \$7,000, display the bonus amount for the employee as 18% of the salary.
- c) When the employee's salary is between \$7,000 and \$15,000, display the bonus amount for the employee as 12% of the salary.
- d) When the employee's salary exceeds \$15,000, display the bonus amount for the employee as 10% of the salary.

- e) When the employee's salary is NULL, display the bonus amount for the employee as 0.
- f) Test the PL/SQL block for each case using the following test cases, and check each bonus amount.

Employee Number	Salary	Resulting Bonus
150	26500	2650
179	12000	1440
165	5000	900

3. Create an EMP table that has the following structure:

Employee Number	Salary	Stars
150	26500	
179	12000	
165	5000	

The Stars field is of variable character type.

4. Create a PL/SQL block that rewards an employee by appending an asterisk in the STARS column for every \$1000 of the employee's salary.
- Use the DEFINE command to provide the employee ID. Pass the value to the PL/SQL block through a /SQL*Plus substitution variable.
 - Initialize a v_asterisk variable that contains a NULL.
 - Append an asterisk to the string for every \$1000 of the salary amount. For example, if the employee has a salary amount of \$8000, the string of asterisks should contain eight asterisks. If the employee has a salary amount of \$12500, the string of asterisks should contain 13 asterisks.
 - Update the STARS column for the employee with the string of asterisks.
 - Commit.
 - Test the block.

5 Declaration and Processing Data: INDEX BY Tables and PL/SQL Record Declaration

Objectives:

- Declare INDEX BY tables.
- Process data by using INDEX BY tables.
- Declare a PL/SQL record.
- Process data by using a PL/SQL record.

Description:

1. Write a PL/SQL block to print information about a given course.
 - a) Declare a PL/SQL record based on the structure of the COURSES table.
 - b) Use the DEFINE command to provide the course ID. Pass the value to the PL/SQL block through an iSQL*Plus substitution variable.
 - c) Use DBMS_OUTPUT.PUT_LINE to print selected information about the country. A sample output is shown below.

Course Id: WBSVR10 Course Title: ORACLE9i Application Server Administration Duration: 2 days

PL/SQL procedure successfully completed.

2. Create a PL/SQL block to retrieve the name of each department from the DEPARTMENTS table and print each department name on the screen, incorporating an INDEX BY table.
 - a) Declare an INDEX BY table, MYDEPTTABLE, to temporarily store the name of the departments.
 - b) Using a loop, retrieve the name of all departments currently in the DEPARTMENTS table and store them in the INDEX BY table. Use the following table to assign the value for DEPARTMENT_ID based on the value of the counter used in the loop.

COUNTER	DEPARTMENT_ID
1	10
2	20
3	50
4	60
5	80
6	90
7	110

6 Explicit Cursors: Querying Rows of a Table Cursor FOR Loop and Cursor Attributes

Objectives:

- Declare and use explicit cursors to query rows of a table.
- Use a cursor FOR loop.
- Apply cursor attributes to test the cursor status.

Description:

1. Create a new table for storing the fees of courses.

```
CREATE TABLE top_courses  
    fee NUMBER(8,2);
```

2. Create a PL/SQL block that determines the top courses with respect to fees.
 - a) Accept a number n from the user where n represents the number of top n courses from the COURSES table. For example, to view the top five courses, enter 5.
 - b) In a loop use the iSQL*Plus substitution parameter created in step 1 and gather the salaries of the top n people from the EMPLOYEES table. There should be no duplication in the fees. If two courses have the same fee, the fee should be picked up only once.
 - c) Store the fees in the TOP_COURSES table.
 - d) Test a variety of special cases, such as n = 0 or where n is greater than the number of courses in the COURSES table. Empty the TOP_COURSES table after each test.

3. Create a PL/SQL block that does the following:
 - a) Uses the DEFINE command to provide the category ID. Pass the value to the PL/SQL block through an iSQL*Plus substitution variable.
 - b) In a PL/SQL block, retrieve the course title, fee, and category ID of the courses.

7 Explicit Cursors – Declaration and Use For Update Cursor

Objectives:

- Declare and use explicit cursors with parameters.
- Use a FOR UPDATE cursor.

Description:

1. In a loop, use a cursor to retrieve the category number and the category name from the CATEGORIES table for those categories whose CATEGORY_ID is less than 600.
2. Pass the category number to another cursor to retrieve from the COURSES table the details of course titles, duration, description, and fees for courses whose FEE is less than \$1250 and which belong to that category.

8 Exception: Types, Trapping, and Handling

Objectives:

- Use exception types
- Use exception trapping.
- Use exception handling.

Description:

1. Write a PL/SQL block to select the name of the employee with a given salary value.
 - a) Use the DEFINE command to provide the salary. Pass the value to the PL/SQL block through an iSQL*Plus substitution variable. If the salary entered returns more than one row, handle the exception with an appropriate exception handler and insert into the MESSAGES table the message "More than one employee with a salary of <salary>."
 - b) If the salary entered does not return any rows, handle the exception with an appropriate exception handler and insert into the MESSAGES table the message "No employee with a salary of <salary>."
 - c) If the salary entered returns only one row, insert into the MESSAGES table the employee's name and the salary amount.
 - d) Handle any other exception with an appropriate exception handler and insert into the MESSAGES table the message "Some other error occurred."
 - e) Test the block for a variety of test cases. Display the rows from the MESSAGES table to check whether the PL/SQL block has executed successfully.

2. Write a PL/SQL block that prints the number of employees which earn plus or minus \$100 of the salary value set for an SQL*Plus substitution variable. Use the DEFINE command to provide the salary value. Pass the value to the PL/SQL block through an iSQL*Plus substitution variable.
 - a) If there is no employee within that salary range, print a message to the user indicating that this is the case. Use an exception for this case.
 - b) If there are one or more employees within that range, the message should indicate how many employees are in that salary range.
 - c) Handle any other exception with an appropriate exception handler. The message should indicate that some other error occurred.

9 Stored Procedures**Objectives:**

- Create stored procedures.
- Handle exceptions in procedures.
- Compile and invoke procedures.

Description:

1. Create and invoke the ADD_CATEGORY procedure and consider the results.
 - a) Create a procedure called ADD_CATEGORY to insert a new category into the CATEGORIES table. Provide the ID and name of the category, using two parameters.
 - b) Compile the code, and invoke the procedure with M110 as category ID and Web Server as category name. Query the CATEGORIES table to view the results.
 - c) Invoke the procedure again, passing a category ID of M120 and a category name of Internet Security.
2. Create a procedure called UPD_CATEGORY to modify a category in the CATEGORIES table.
 - a) Create a procedure called UPD_CATEGORY to update the category name. Provide the category ID and a new name, using two parameters. Include the necessary exception handling if no update occurs.
 - b) Compile the code; invoke the procedure to change the category ID M110 to Web Server Security. Query the CATEGORIES table to view the results.

3. Create a procedure called DEL_CATEGORY to delete a category from the CATEGORIES table.
 - a) Create a procedure called DEL_CATEGORY to delete a job. Include the necessary exception handling when no category is deleted.
 - b) Compile the code; invoke the procedure using category ID M120. Query the CATEGORIES table to view the results.

10 Stored Functions

Objectives:

- Create stored functions.
- Invoke a stored function from a SQL statement.
- Invoke a stored function from a stored procedure.

Description:

1. Create and invoke the Q_CATEGORY function to return a category name.
 - a) Create a function called Q_CATEGORY to return a category name to a host variable.
 - b) Compile the code; create a host variable G_NAME and invoke the function with category ID M110. Query the host variable to view the result.
2. Create a function called PER_DAY_FEE to return the per day fee by accepting two parameters -- the course fee and the duration of that course in days. The function should address NULL values.
 - a) Create and invoke the function PER_DAY_FEE, passing in values for course fee and its duration in number of days. Either or both values passed can be NULL, but the function should still return the per day fee, which is not NULL.

The total fee is defined by the basic formula:

$$\text{fee/duration}$$

- b) Use the function in a SELECT statement against the COURSE table for category ID M090.

3. Create a procedure, `NEW_COURSE`, to insert a new course into the `COURSES` table. The procedure should contain a call to the `VALID_CATEGORY_ID` function to check whether the category ID specified for the new course exists in the `CATEGORIES` table.
 - a) Create the function `VALID_CATEGORY_ID` to validate a specified category ID. The function should return a `BOOLEAN` value.
 - b) Create the procedure `NEW_COURSE` to add a course to the `COURSES` table. A new row should be added to the `COURSES` table when the function returns `TRUE`. If the function returns `FALSE`, the procedure should alert the user with an appropriate message.
 - c) Test this `NEW_COURSE` procedure by adding a new course named `ORACLE Application Server` to category id `M110`.

11 Packages

Objectives:

- Create a package.
- Invoke package program units.

Description:

1. Create a package specification and body called CATEGORIES_PACK. (You can save the package body and specification in two separate files.) This package contains your ADD_CATEGORY, UPD_CATEGORY, and DEL_CATEGORY procedures, as well as your Q_CATEGORY function.

Use the code from the saved script files when creating the package.

- a) Make all the constructs public.

Consider whether you still need the stand-alone procedures and functions you just packaged.

- b) Invoke the ADD_CATEGORY procedure by passing values M130 and Server-side Programming as parameters.
 - c) Query the JOBS table to see the result.
2. Create and invoke a package that contains private and public constructs.
 - a) Create a package specification and package body called COURSE_PACK that contains the NEW_COURSE procedure as a public construct, and the VALID_CATEGORY_ID function as a private construct.

- b) Invoke the NEW COURSE procedure, using M095 as a category number. If the category ID M095 does not exist in the CATEGORIES table, there will be an error message as specified in the exception handler of the procedure.

- c) Invoke the NEW_COURSE procedure, using an existing category ID M100.

