

CODE PL/SQL SUBPROGRAMS

SECTION OBJECTIVES

At the end of this section, you should be able to:

- 1 Understand the advantages of writing subprograms.
- 2 Explain the differences between PL/SQL procedures and functions.
- 3 Create and invoke a PL/SQL procedure.

CODE PL/SQL SUBPROGRAMS: OVERVIEW

DECLARE

```
...  
PROCEDURE... IS  
  
BEGIN  
  
[EXCEPTION]  
  
END;  
...
```

BEGIN

```
...  
procedure_name...;  
...
```

Code PL/SQL Subprograms: Overview—cont'd

Aid application development by writing PL/SQL subprograms (procedures and functions).

PL/SQL Subprograms

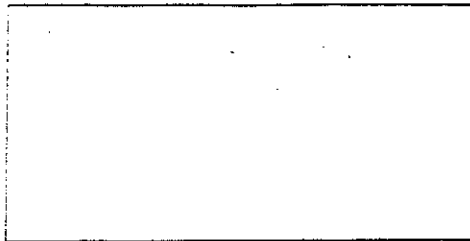
- Provide extensibility
- Provide modularity
- Promote reusability
- Promote maintainability
- Aid abstraction

Quick Notes

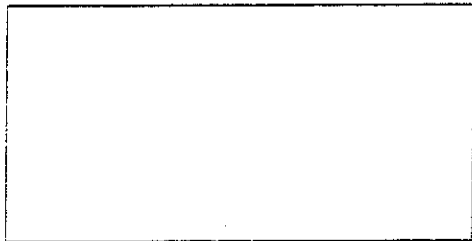
- Incorporate a subprogram in a PL/SQL block by coding the subprogram in the declaration section for that block.
- Procedures and functions contain a declaration section, an executable section, and an optional exception handling section.

CREATE A PL/SQL PROCEDURE

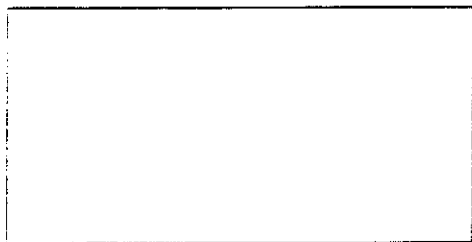
PROCEDURE ... IS



BEGIN



[EXCEPTION]



END;

Create a PL/SQL Procedure—cont'd

A PL/SQL procedure is a named, callable block that performs a specific action.

Syntax—Procedure body

```
PROCEDURE procedure_name [ (parameter [, parameter, . . .] ) ] IS
  [local declarations]
BEGIN
  executable statements
  [EXCEPTION
  exception handlers]
END [procedure_name];
```

where: *parameter* represents the following:
parameter_name mode datatype [:= *value*]

Syntax—Call a procedure

```
procedure_name [ (parameter_name [, parameter_name, . . .] ) ]
```

Quick Note

- PL/SQL procedures are specific to ORACLE7.

Create a PL/SQL Procedure—cont'd

Pass information between procedures using parameters.

Syntax - Use parameters

```
PROCEDURE procedure_name (parameter_name mode datatype [:=value])
```

where:	IN	allows for passing values in to a procedure. It acts as a constant and cannot be assigned a value in the procedure. IN is the default mode.
	OUT	allows for returning values back to the caller of the procedure. It acts as an un-initialized variable and cannot be assigned to another variable or reassigned to itself in the procedure.
	IN OUT	allows for passing initial values in to a procedure and returning updated values to the caller. It acts as an initialized variable.

Create a PL/SQL Procedure—cont'd

Example - Call a procedure with IN parameters

```
raise_salary (emp_num, amount);
```

Example - Procedure body with IN parameters

```
PROCEDURE raise_salary (emp_id IN INTEGER, increase IN REAL) IS
    current_salary REAL;
    salary_missing EXCEPTION;
BEGIN
    SELECT sal INTO current_salary FROM emp
        WHERE empno = emp_id;
    IF current_salary IS NULL THEN
        RAISE salary_missing;
    ELSE
        UPDATE emp SET sal = sal + increase
            WHERE empno = emp_id;
    END IF;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        INSERT INTO emp_audit VALUES (emp_id, 'No such number');
    WHEN salary_missing THEN
        INSERT INTO emp_audit VALUES (emp_id, 'Salary is null');
END raise_salary;
```

Create a PL/SQL Procedure—cont'd

Example - Call a procedure with an IN and an OUT parameter

```
calc_bonus (employee_num, bonus_amt);
```

Example - Procedure body with an IN and an OUT parameter

```
PROCEDURE calc_bonus (emp_id IN INTEGER, bonus OUT REAL) IS
    hire_date DATE;
    commission NUMBER (7,2);
BEGIN
    SELECT hiredate, comm into hire_date, commission FROM emp
        WHERE empno = emp_id;
    IF MONTHS_BETWEEN (SYSDATE, hire_date) > 60 THEN
        bonus := commission + 500;
    END IF;
END calc_bonus;
```

Create a PL/SQL Procedure—cont'd

Example - Call a procedure with an IN and an IN OUT parameter

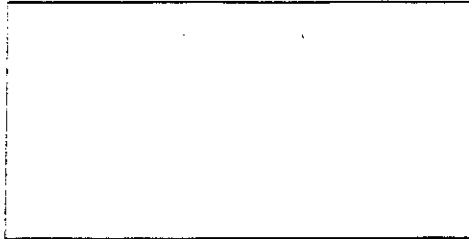
```
debit_account (account_num, purchase);
```

Example - Procedure body with an IN and an IN OUT parameter

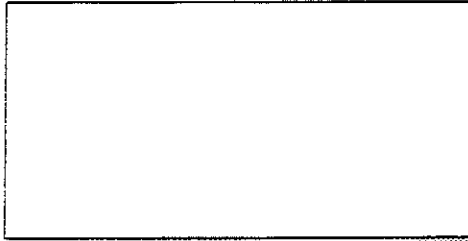
```
PROCEDURE debit_account (acct_id IN INTEGER, amount IN OUT REAL) IS  
    minimum_purchase CONSTANT REAL := 10.5;  
    service_charge CONSTANT REAL := 0.50;  
BEGIN  
    IF amount < minimum_purchase THEN  
        amount := amount + service_charge;  
    END IF;  
    . . .  
END debit_account;
```

CREATE A PL/SQL FUNCTION

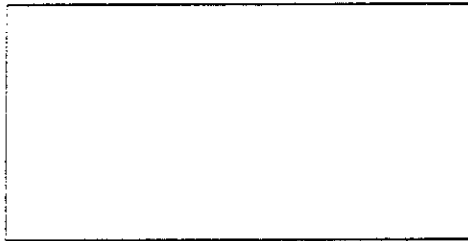
FUNCTION ... IS



BEGIN



[EXCEPTION]



END;

Create a PL/SQL Function—cont'd

A PL/SQL function is a named, callable block that performs a specific action and returns a value. Functions are structured the same as procedures except functions have a RETURN clause.

Syntax - function body

```
FUNCTION function_name [(parameter [, parameter, ...])] RETURN datatype IS
  [local declarations]
BEGIN
  executable statements
  [EXCEPTION
  exception handlers]
END [function_name];
```

where: *parameter* represents the following:
parameter_name [IN] datatype [:= value]

Syntax - Call a function as part of an expression

```
.... function_name [ (parameter_name [, parameter_name, ...] ) ] ...
```

Quick Notes

- Functions are specific to ORACLE7.
- A function can only take IN parameters.
- A function must be called as part of an expression.
- Calls to user-defined functions can appear in procedural statements, but not in SQL statements. *FEEL*

*select mm-funk (ename), sal
from ...*

Create a PL/SQL Function—cont'd

Example - Call a function as part of an expression

```
IF sal_ok (new_sal, level) THEN
. . .
END IF;
```

Or

```
promotable := sal_ok (new_sal, level) AND (rating > 3);
```

Example - Function body

```
FUNCTION sal_ok (salary IN REAL, level IN CHAR) RETURN BOOLEAN IS
min_sal REAL;
max_sal REAL;
BEGIN
SELECT losal, hisal INTO min_sal, max_sal FROM salgrade
WHERE grade = level;
RETURN (salary >= min_sal) AND (salary <= max_sal);
END sal_ok;
```

Create a PL/SQL Function—cont'd

One or more RETURN statements must appear in the executable part of a function.

The RETURN Statement

- A function can contain several RETURN statements, none of which need to be the last lexical statement.
- The RETURN statement must contain an expression, which is evaluated when the statement is executed. The resulting value of the expression is assigned to the function identifier.
- The RETURN statement returns control to the point where the function was called. Execution then resumes with the statement following the function call.

Create a PL/SQL Function—cont'd

One or more RETURN statements must appear in the executable part of a function.

The RETURN Statement

- A function can contain several RETURN statements, none of which need to be the last lexical statement.
- The RETURN statement must contain an expression, which is evaluated when the statement is executed. The resulting value of the expression is assigned to the function identifier.
- The RETURN statement returns control to the point where the function was called. Execution then resumes with the statement following the function call.

Code PL/SQL Subprograms: Summary—cont'd

Use PL/SQL subprograms as building blocks to develop reliable, modular, and maintainable applications.

PL/SQL Subprograms

- A PL/SQL procedure is a named, callable block that performs a specific action.
- A PL/SQL function is a named, callable block that performs a specific action and returns a value.

LAB 6-1

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	

