3

CODE CONDITIONAL AND ITERATIVE CONTROL

SECTION OBJECTIVES

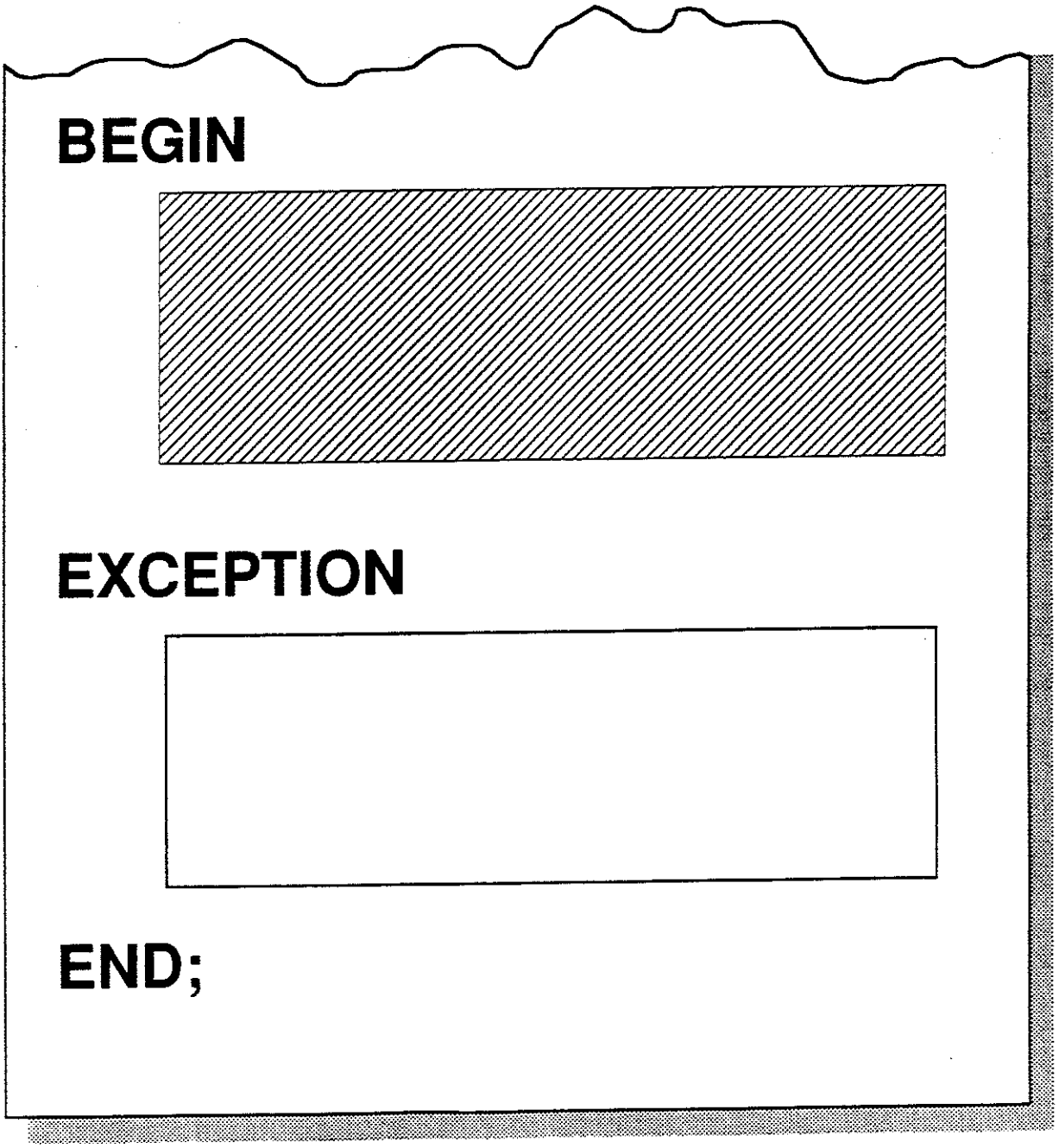
At the end of this section, you should be able to:

1. Demonstrate an understanding of PL/SQL conditional control by writing a PL/SQL procedure to conditionally execute SQL statements.
2. Identify the four types of loops supported by PL/ SQL.
3. Demonstrate an understanding of PL/SQL iterative control by writing a simple PL/SQL procedure using a loop structure.
4. Differentiate between legal and illegal GOTO statements.
5. Identify three uses for PL/SQL statement labels.

CODE CONDITIONAL AND ITERATIVE CONTROL: OVERVIEW

Code conditional and iterative control statements within a PL/SQL block.

- Perform a statement or a sequence of statements depending on the result of a logical comparison.
- Repeat a statement or sequence of statements multiple times with a loop.
- Utilize statement labels.



PERFORM LOGICAL COMPARISONS

Compare variables and constants in both SQL and procedural statements.

Logical Comparisons

- Consist of simple or complex expressions separated by relational operators.
- Results are always either TRUE, FALSE, or NULL.

Null Comparisons

- Anything compared with NULL results in a NULL value.
- A NULL in an expression evaluates to NULL (except concatenation).

Examples

```
5 + NULL -- evaluates to NULL
```

```
'PL/' || NULL || 'SQL' -- evaluates to PL/SQL
```

Perform Logical Comparisons—cont'd

Logical comparisons often contain Boolean expressions connected by logical operators: AND, OR, or NOT.

AND	TRUE	FALSE	NULL
TRUE	T	F	N
FALSE	F	F	F
NULL	N	F	N

OR	TRUE	FALSE	NULL
TRUE	T	T	T
FALSE	T	F	N
NULL	T	N	N

NOT	
TRUE	F
FALSE	T
NULL	N

CODE IF-THEN-ELSE STATEMENTS

Conditionally execute a statement or sequence of statements with the IF-THEN-ELSE statement.

Syntax - IF-THEN-ELSE statement

```
IF <condition> THEN
     <sequence of statements>
[ELSIF <condition> THEN
     <sequence of statements>]
-- ELSIFs can be repeated
[ELSE
     <sequence of statements>]
END IF;
```

Quick Notes

- The *<condition>* must evaluate to a Boolean datatype (TRUE, FALSE, or NULL)
- If *<condition>* is TRUE, then the associated <sequence of statements> is executed; otherwise, it is not.
- At most one <sequence of statements> gets executed.

Code IF-THEN-ELSE Statements—cont'd

Example - IF-THEN-ELSE statement

```
DECLARE
num_jobs        NUMBER (7);

BEGIN
SELECT COUNT(*) INTO num_jobs FROM auditions
        WHERE actorid = &&actor_id AND called_back = 'YES';

IF num_jobs > 90 THEN
     UPDATE actor SET actor_rating = 'OSCAR time'
        WHERE actorid = &&actor_id;

ELSIF num_jobs > 75 THEN
     UPDATE actor SET actor_rating = 'Daytime soaps'
        WHERE actorid = &&actor_id;

ELSE
     UPDATE actor SET actor_rating = 'Waiter'
        WHERE actorid = &&actor_id;
END IF;

COMMIT;
END;
```

Code IF-THEN-ELSE Statements—cont'd

Avoid the NULL trap when coding IF-THEN-ELSE statements.

BLOCK 1:

```
.
.
.
IF a >= b THEN
    do_this…;
ELSE
    do_that…;
END IF;
```

BLOCK 2:

```
.
.
.
IF b > a THEN
     do_that…;
ELSE
     do_this…;
END IF;
```

- Given any pair of non-NULL values for *a* and *b*, will block 1 and block 2 do the same thing?

- What if either *a* or *b* (or both) is NULL?

EDIT USING THE VAX EDITOR

1 To edit a file enter: `ed` *filename* <Return>

2 Edit in screen mode

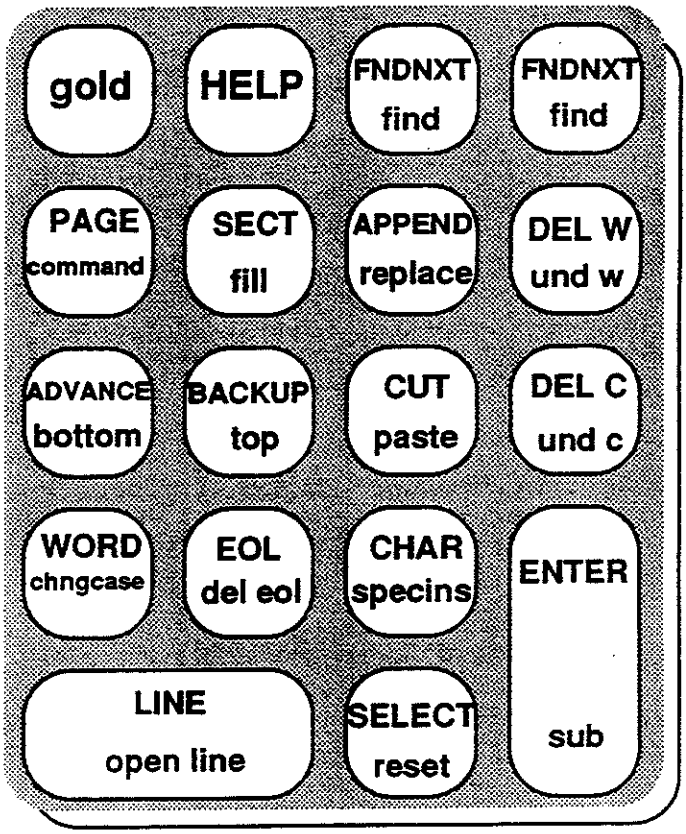
3 To exit screen mode enter: <Ctrl><Z>

4 To write out file and exit enter: `ex`<Return>

To exit without writing out file enter: `quit`<Return>

Screen Mode Commands

Command	Description
UP	Move up
LEFT	Move left
RIGHT	Move right
DELETE	Delete character
CTRL/R	Refresh screen
CTRL/Z	Exit to line mode



EDIT USING THE UNIX EDITOR

1 To edit a file enter: `edit` *filename* <Return>

2 To input information, press the appropriate keys in the Input Mode list.

3 To exit Input Mode, press <Esc>.

4 To edit existing characters, press the appropriate key in the Edit Mode list.

5 To write out file and exit enter: `ZZ`

To exit without writing out the file enter: `:q!`

To exit, name, and write out the file enter: `:w` *filename*

Input Mode

i	Insert before cursor position
I	Insert at beginning of line
a	Append after cursor position
A	Append at end of line
o	Open a blank line below cursor position
O	Open a blank line above cursor position

Edit Mode

x	Delete the character at cursor position
dd	Delete this line
r	Replace the character at cursor position
R	Replace all subsequent characters until <Esc> is pressed
u	Undo the most recent change
U	Undo the most recently deleted text

LAB 3-1

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

Lab 3-1—cont'd

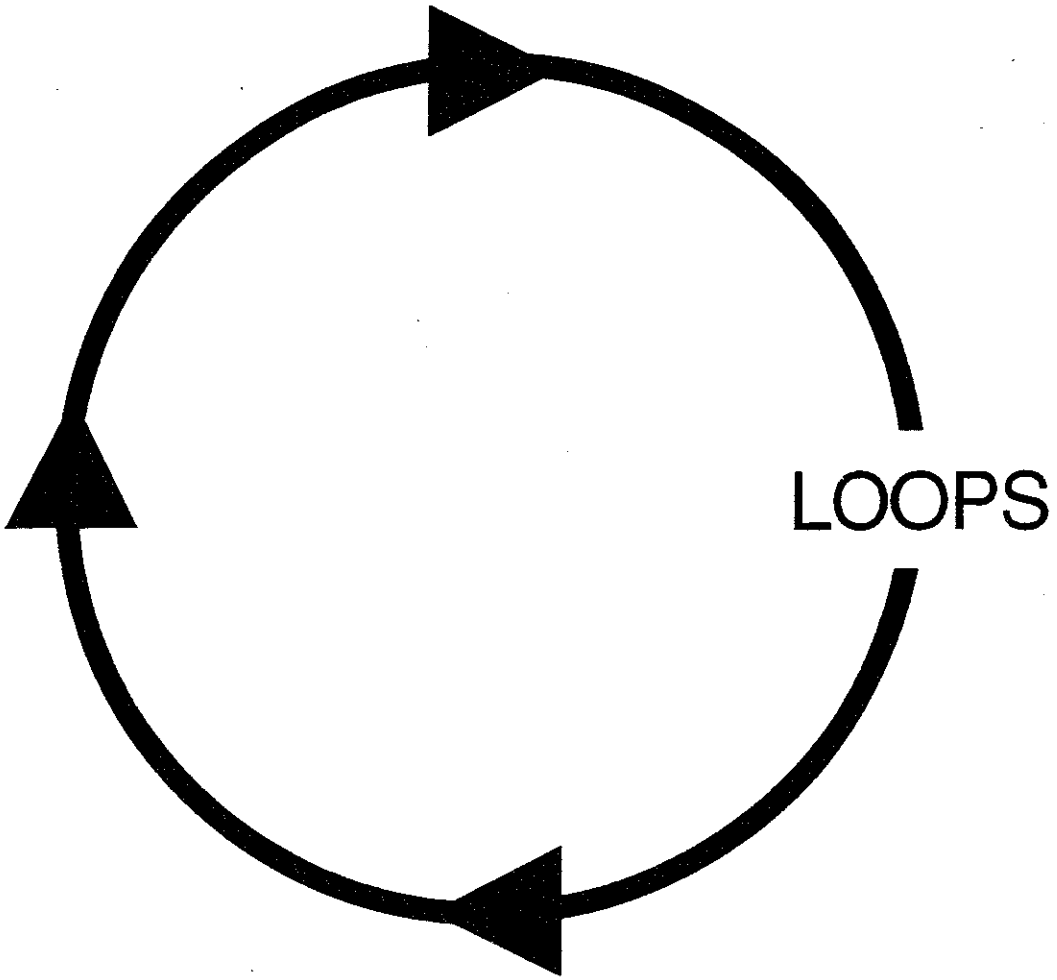
CUST_ORDERS TABLE

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

SPECIAL_ORDERS TABLE

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

LOOP STATEMENT OVERVIEW



Loop Statement Overview—cont'd

Repeat a statement or sequence of statements multiple times with a loop.

Types of Loops

- Simple Loops
- Numeric FOR Loops
- WHILE Loops
- Cursor FOR Loops

CODE SIMPLE LOOPS

Repeat a sequence of statements multiple times with a simple loop.

Syntax - Simple loop

```
LOOP
     <sequence of statements>
END LOOP;      -- sometimes called an 'infinite' loop
```

Syntax - Exit

Exit any type of loop immediately with an exit statement.

```
EXIT [WHEN <condition> ]; -- 'infinite' loop insurance
```

Code Simple Loops—cont'd

Examples

```
DECLARE
ctr     NUMBER(3) := 0;
BEGIN
LOOP
     INSERT INTO LOG VALUES
           (ctr,'ITERATION COMPLETE');
     ctr := ctr + 1;
     IF ctr = 500 THEN
           EXIT;
     END IF;
END LOOP;
END;
```

```
DECLARE
ctr     NUMBER(3) := 0;
BEGIN
LOOP
     UPDATE TABLE1 SET comment = 'UPDATED'
        WHERE count_col = ctr;
     ctr := ctr + 1;
     EXIT WHEN ctr = 100;
END LOOP;
END;
```

CODE NUMERIC FOR LOOPS

Repeat a sequence of statements a fixed number of times with a numeric FOR loop.

Syntax - Numeric FOR loop

```
FOR <index> IN [ REVERSE ] <integer>..<integer> LOOP
     <sequence of statements>
END LOOP
```

The loop index takes on each value in the range, one at a time, either in forward or reverse order.

Example

```
BEGIN
     FOR i IN 1..500 LOOP
        INSERT INTO temp (col1,message)
           VALUES (i, 'I will not sleep in class.');
     END LOOP;
END;
```

se 3-22

Code Numeric FOR Loops—cont'd

Examples

```
DECLARE
my_index      CHAR(20) := 'Fettucini Alfredo';
BEGIN
FOR my_index IN REVERSE 21..30 LOOP
     /* redeclares my_index */
        INSERT INTO temp (col1)
           VALUES (my_index);
           /* inserts the numbers 30 through 21 */
END LOOP;
END;
```

```
...
FOR i IN 1..256 LOOP
     x := x + i; -- legal
     i := i + 5; -- illegal
END LOOP;
...
```

Quick Notes

- It is implicitly of type NUMBER.
- It is only defined within the loop.
- The value can be referenced in an expression, but a new value can not be assigned to the index within the loop

else for handling the pi getter i:

if mod (i, 2) = 0 then ...

CODE WHILE LOOPS

Repeat a sequence of statements until a specific condition is no longer TRUE with the WHILE loop.

Syntax - WHILE loop

```
WHILE <condition> LOOP
     <sequence of statements>
END LOOP;
```

Example

```
DECLARE
ctr      NUMBER(3) := 0;
BEGIN
WHILE ctr < 500 LOOP
     INSERT INTO temp (col1,message)
        VALUES  (ctr, 'Well, I might sleep just a little.');
     ctr := ctr + 1;
END LOOP;
END;
```

Quick Notes

- The term <*condition*> can be any legal PL/SQL condition (in other words, it must return a BOOLEAN value of TRUE, FALSE, or NULL).
- The sequence of statements will be repeated as long as <*condition*> evaluates to TRUE.

CODE GOTO STATEMENTS

Transfer control to a different place in the PL/SQL block with a GOTO statement.

GOTO Statement Parts

- The GOTO statement itself
- A statement label

Syntax - GOTO Statement

```
<<label_name>> x := x + 1;          -- a statement label
GOTO label_name;                    -- transfers control to x := x + 1
```

Code GOTO Statements—cont'd

Not all GOTOs are legal.

Legally use a GOTO to transfer control to a statement that is :

- in the same sequence of statements as the GOTO statement.
- in the sequence of statements that encloses the GOTO statement (in other words, an outer block).

Legal	*Illegal*

<<dinner>>
x := x + 1;
y := y + x;
IF a > b THEN
b := b + c;
GOTO dinner;
END IF;
	GOTO your_brothers;
	IF a > b THEN
	b := b - c;
	<<your_brothers>>
	x := x + 1;
	END IF;

REFERENCE STATEMENT LABELS

Labels can label any statement.

- In addition to their use as targets for GOTO statements, use labels for blocks and loops.
- Label a block to allow referencing of DECLAREd objects that would otherwise not be visible because of scoping rules.

Syntax

```
<<label_name>>
[ DECLARE
     -- declarations go here ]
BEGIN
     -- executable statements go here
[ EXCEPTION
     --exception handlers go here ]
END label_name;  -- must include the label_name
```

Reference Statement Labels—cont'd

Example

```
<<outer_block>>
DECLARE
  n     NUMBER;
BEGIN
n := 5;
        /* start a sub-block */
     DECLARE
        x     NUMBER := 10;
        n     CHAR(10) := 'Fifteen';
     BEGIN
           INSERT INTO temp (col1,col2,message)
              VALUES (outer_block.n, x, n);
           COMMIT;
     END;  /* end of the sub-block */
END outer_block;
```

Reference Statement Labels—cont'd

Label a block to allow a variable to be referenced that might be hidden by a column name.

Example

```
<<sample>>
DECLARE
     deptno      NUMBER := 20;
BEGIN
     UPDATE emp SET sal = sal * 1.1
        WHERE deptno = sample.deptno;
     COMMIT;
END sample;
```

Reference Statement Labels—cont'd

Label a loop to allow an object to be referenced that would otherwise not be visible because of scoping rules.

Example

```
<<compute_loop>>
FOR i IN 1..10 LOOP
     <statements …>
     DECLARE
          i     NUMBER := 0;
     BEGIN
        INSERT INTO temp (col1,col2,message)
           VALUES (i, compute_loop.i, 'COMPLETE');
     END;
END LOOP compute_loop; -- must include loop name here
```

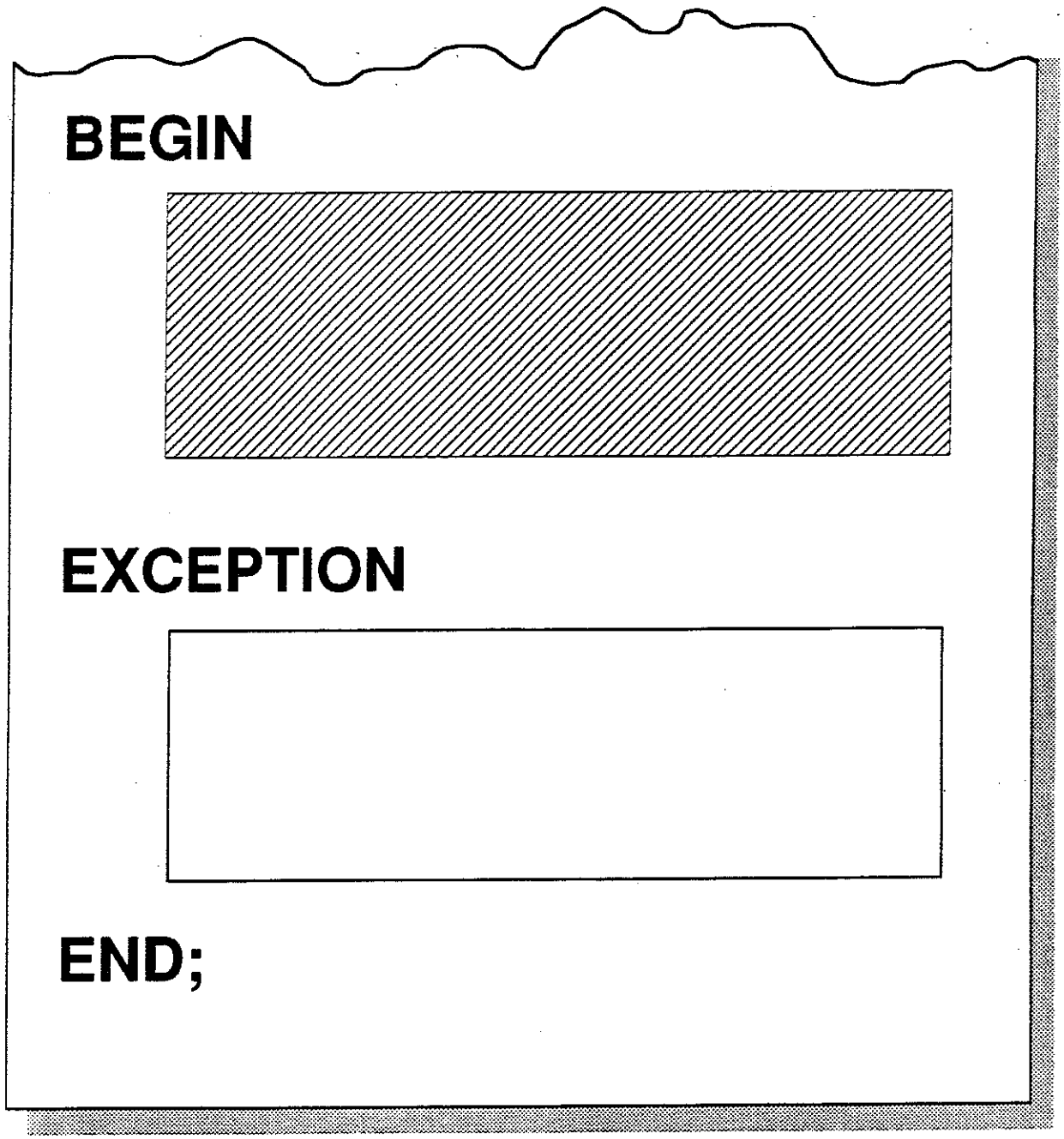
Reference Statement Labels—cont'd

Label an exit as a convenient way to specify exits from outer loops.

Example

```
...
<<outer_loop>> WHILE a > b LOOP
b := b + 1;
<<inner_loop>> WHILE b > c LOOP
        c := c + 2;
        EXIT outer_loop WHEN c > 200;
END LOOP inner_loop;
END LOOP outer_loop;
...
```

CODE CONDITIONAL AND ITERATIVE CONTROL: SUMMARY



Code Conditional and Iterative Control: Summary—cont'd

Code conditional and iterative control statements within PL/SQL.

Perform a statement or a sequence of statements depending on the result of a logical comparison.

- Write IF-THEN-ELSE statements.
- Utilize logical operators—AND, OR, and NOT.

Repeat a statement or sequence of statements multiple times with a loop.

- Simple loops
- Numeric FOR loops
- WHILE loops
- Cursor FOR loops

Quick Note

- Utilize statement labels.

LAB 3-2

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
     --------      --------  ---------
```