5

HANDLE PL/SQL ERRORS

SECTION OBJECTIVES

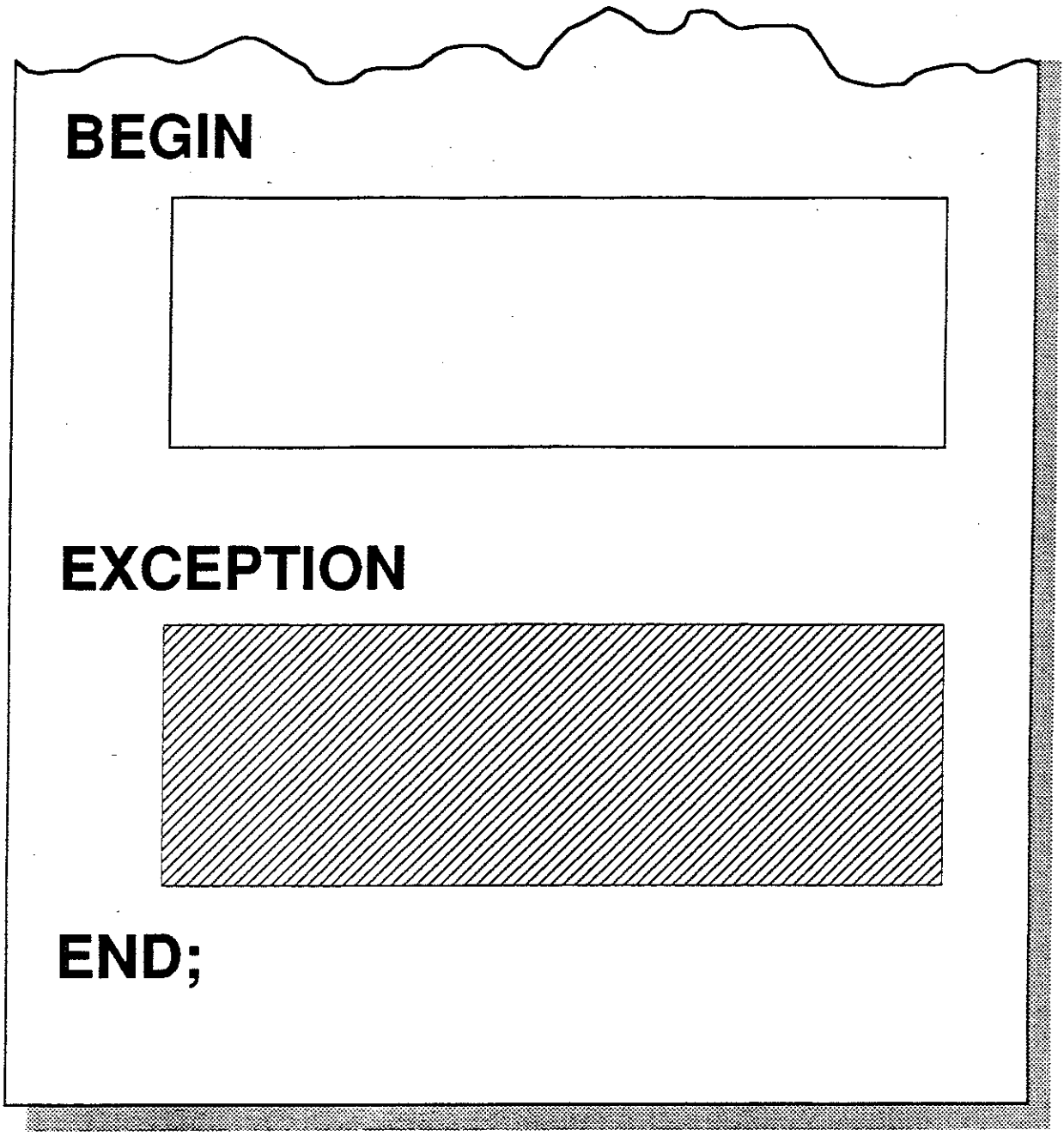
At the end of this section, you should be able to:

1 Identify the two types of exception handling within PL/SQL.

2 Identify the functions supplied by PL/SQL that provide error information.

3 Write a PL/SQL block containing PL/SQL exception handling features that satisfies a given scenario.

HANDLE PL/SQL ERRORS: OVERVIEW



Handle PL/SQL Errors: Overview—cont'd

In PL/SQL, errors are called exceptions. When an exception is raised, processing jumps to the exception handlers.

An exception handler is a sequence of statements to be processed when a certain exception occurs. When an exception handler is complete, processing of the block terminates

Types of Exceptions

- Pre-defined Internal Exceptions

  Correspond to approximately 20 common ORACLE errors

  Raised automatically by PL/SQL in response to an ORACLE error
- User-defined Exceptions

  Must be declared

  Must be RAISEd explicitly

For further information on the subject see:

Technical Reference Appendix, pages 3 through 7

PL/SQL User's Guide and Reference Version 1.0, 800-20

PREDEFINED INTERNAL EXCEPTIONS

Any ORACLE error raises an exception automatically; some of the more common ones have names.

Examples

```
TOO_MANY_ROWS        (ORA-01427)
- a single row SELECT returned more than one row

NO_DATA_FOUND        (ORA-01403)
- a single row SELECT returned no data

INVALID_CURSOR       (ORA-01001)
- an illegal cursor operation occurred

VALUE_ERROR          (ORA-06502)
- arithmetic, conversion, truncation, or constraint error
occurred

INVALID_NUMBER       (ORA-01722)
- conversion of a character string to a number fails in a SQL
   statement

ZERO_DIVIDE          (ORA-01476)
- attempted to divide by zero

DUP_VAL_ON_INDEX     (ORA-00001)
- attempted to insert a duplicate value into a column that has a
   unique index specified

CURSOR_ALREADY_OPEN  (ORA-06511)
- attempted to open an already open cursor
```

OTHERS
- tester på andre feil.

DECLARE EXCEPTION HANDLERS

Syntax

```
WHEN  <exception name> [ OR <exception name> ... ] THEN
        <sequence of statements>
 ...
 [WHEN OTHERS THEN      -- if used, must be last handler
    <sequence of statements>]
```

Example

```
DECLARE
employee_num  emp.empno%TYPE;

BEGIN
SELECT empno INTO employee_num FROM emp
        WHERE ename = 'BLAKE';
INSERT INTO temp (col1, message)
        VALUES (employee_num, 'Blake''s employee number.');
DELETE FROM emp WHERE ename = 'BLAKE';

EXCEPTION
WHEN NO_DATA_FOUND THEN
        ROLLBACK;
        INSERT INTO temp (message)
           VALUES('BLAKE not found.');
        COMMIT;
WHEN TOO_MANY_ROWS THEN
        ROLLBACK;
        INSERT INTO temp (message)
           VALUES ('More than one BLAKE found.');
        COMMIT;
WHEN OTHERS THEN
        ROLLBACK;
END;
```

DECLARE USER-DEFINED EXCEPTIONS

Define and explicitly raise User-defined exceptions.

Examples

```
DECLARE
    x                NUMBER;
    my_exception     EXCEPTION;  -- a new object type
    ...
```

RAISE your exception

```
RAISE my_exception;
```

Quick Notes

- Once an exception is RAISEd manually, it is treated exactly the same as if it were a predefined internal exception.
- Declared exceptions are scoped just like variables.
- A user-defined exception is checked for manually and then RAISEd, if appropriate.

Declare User-defined Exceptions—cont'd

Example

```
DECLARE
     my_ename              emp.ename%TYPE := 'BLAKE';
     assigned_projects     NUMBER;
     too_few_projects      EXCEPTION;

BEGIN
     -- get number of projects assigned to BLAKE
     ...
     IF assigned_projects < 3 THEN
        RAISE too_few_projects;
     END IF;

EXCEPTION   -- begin the exception handlers
     WHEN too_few_projects THEN
        INSERT INTO temp
           VALUES (my_ename, assigned_projects,
                'LESS THAN 3 PROJECTS!');
        COMMIT;
     ...
END;
```

EXCEPTION PROPAGATION

Propagation Steps

1 The current block is searched for a handler. If not found, go to step 2.

2 If an enclosing block is found, it is searched for a handler.

3 Steps 1 and 2 are repeated until either there are no more enclosing blocks, or a handler is found.

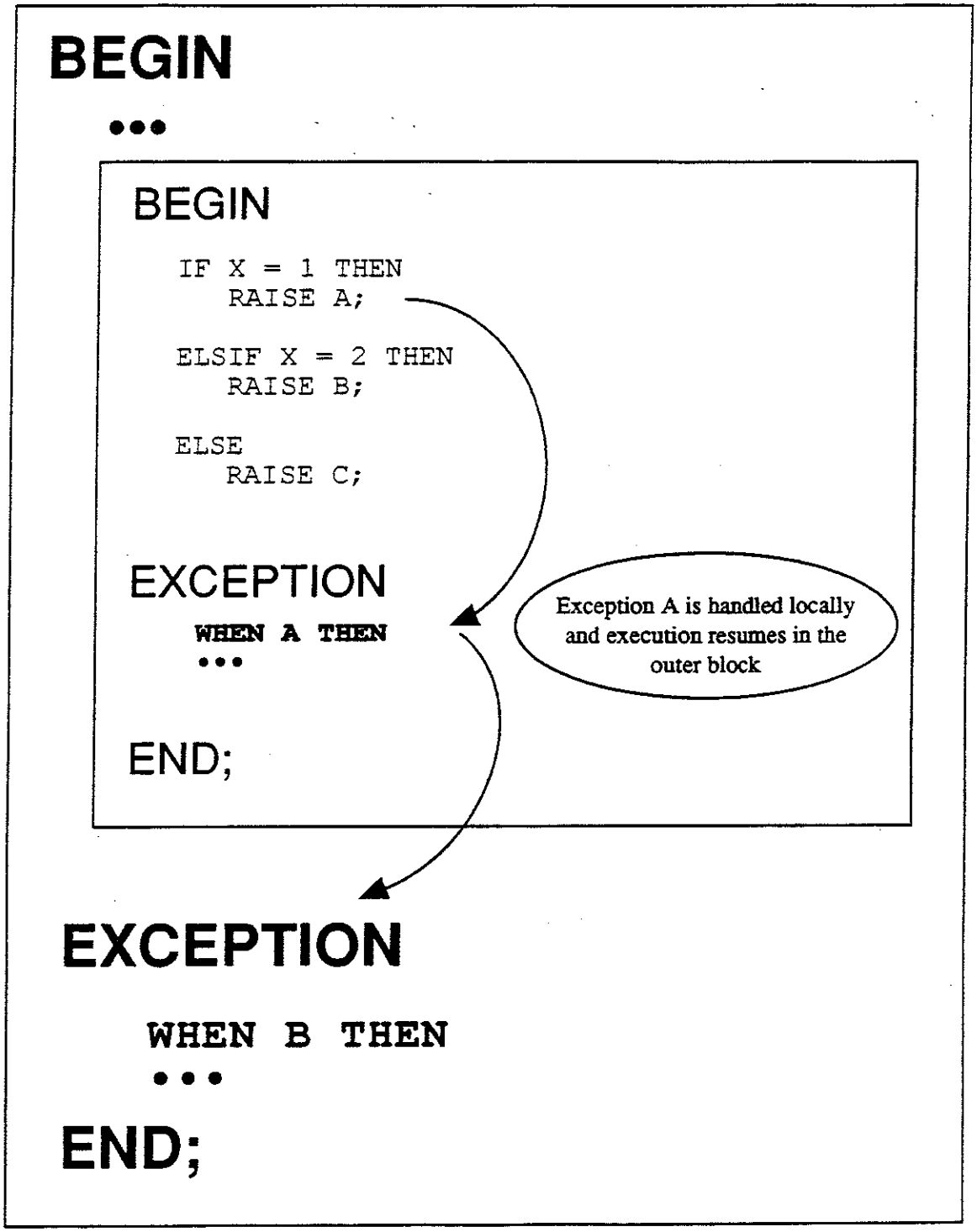
- If there are no more enclosing blocks, the exception is passed back to the calling environment (SQL*Plus, SQL*Forms, a pre-compiled program, and so on).
- If a handler is found, it is executed. When done, the block in which the handler was found is terminated, and control is passed to the enclosing block (if one exists), or to the environment (if there is no enclosing block).

Quick Notes

- Only one handler per block may be active at a time.
- If an exception is raised in a handler, the search for a handler for the new exception begins in the enclosing block of the current block.

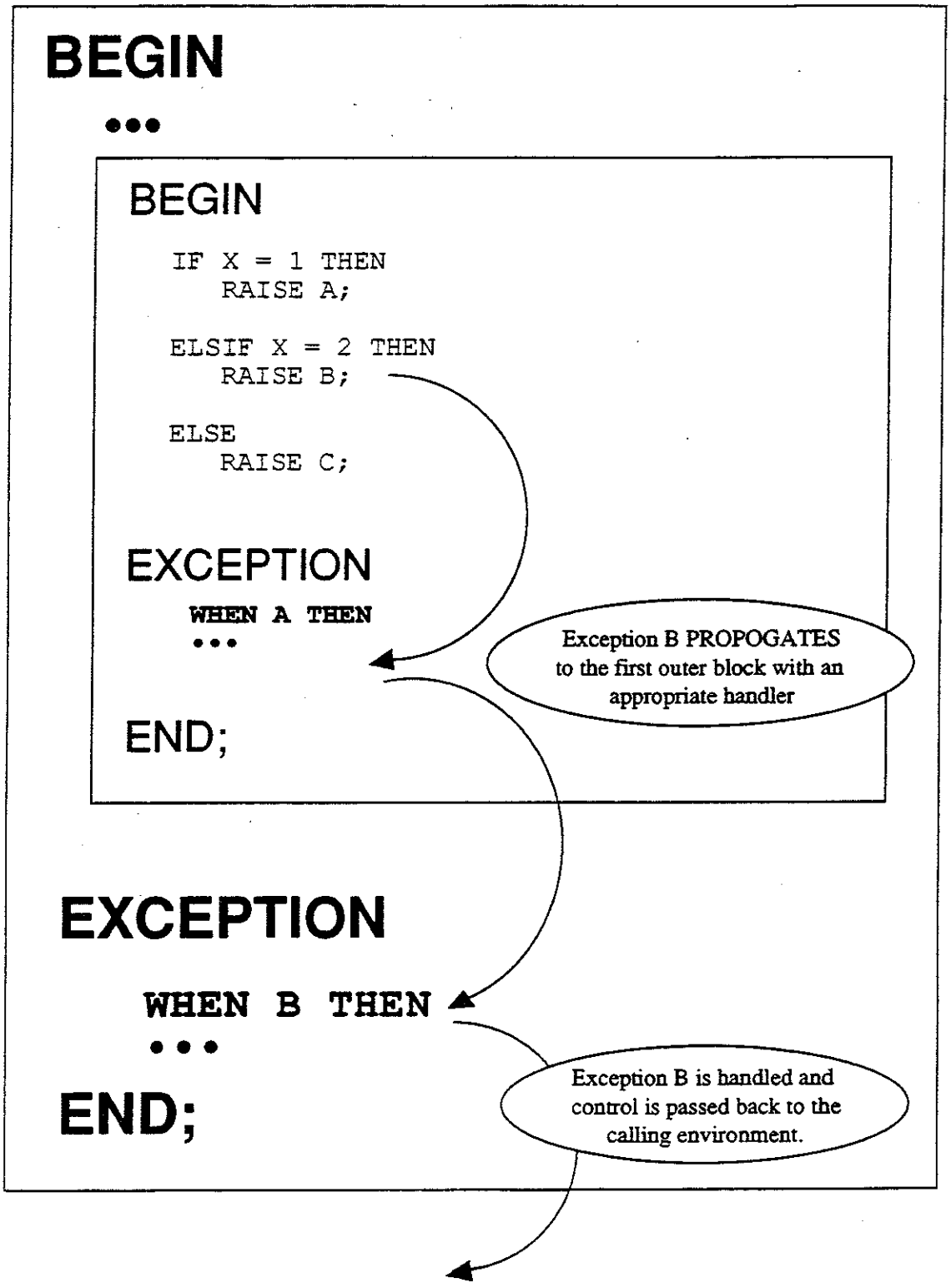
Exception Propagation—cont'd

Example



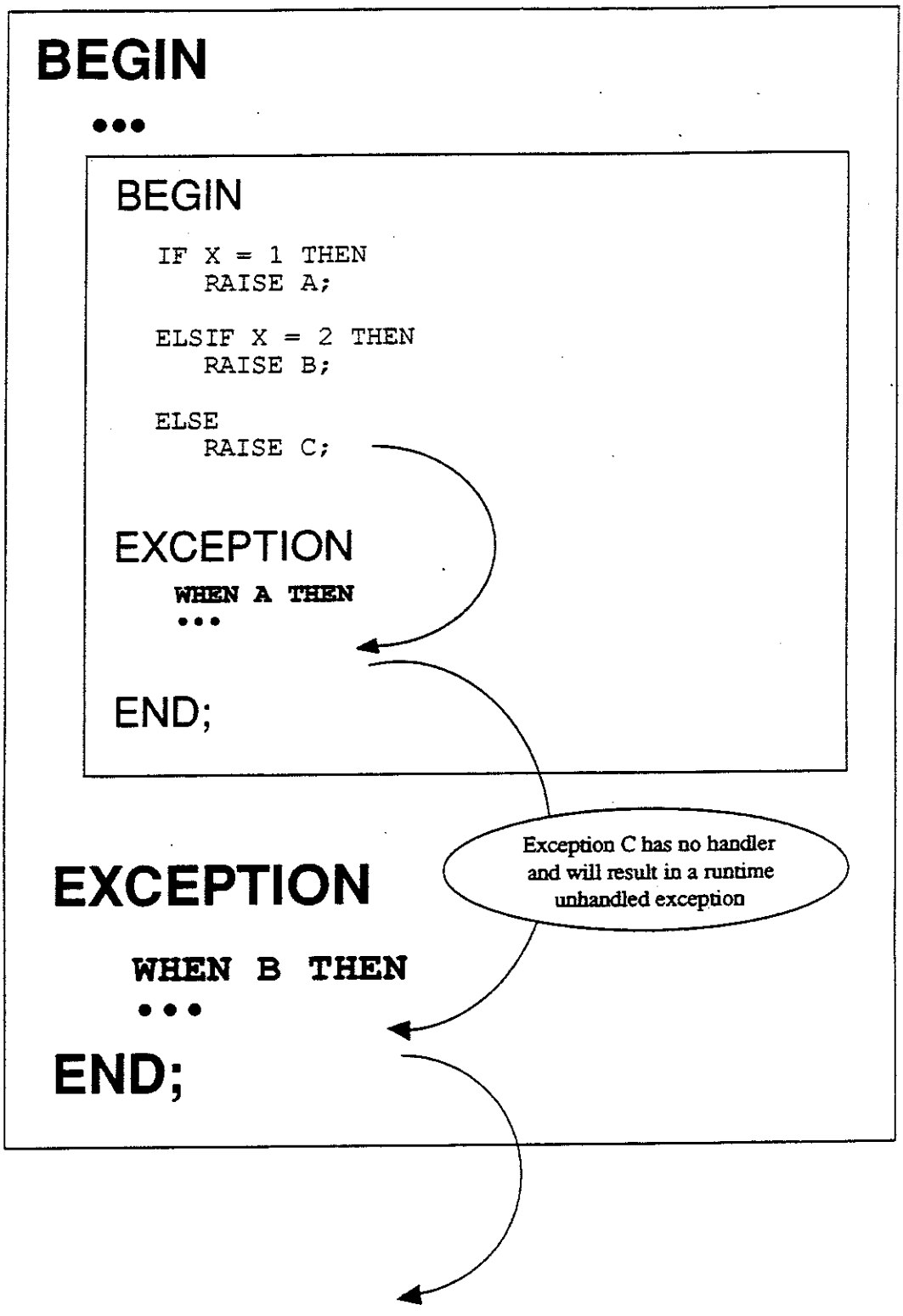
Exception Propagation—cont'd

Example



Exception Propagation—cont'd

Example



OTHER USES OF RAISE

By itself, the RAISE statement simply re-raises the current exception (as if it were being propagated).

Syntax

```
RAISE;
```

Quick Note

- RAISE may only be used in an exception handler.

NAME AN ORACLE ERROR

Exceptions may only be handled by name (not ORACLE error number).

Name an ORACLE error so that a handler can be provided specifically for that error, with EXCEPTION_INIT.

Syntax

```
PRAGMA EXCEPTION_INIT(<user_defined_exception_name>,
                      <ORACLE_error_number>);
```

Example

```
DECLARE
      deadlock_detected    EXCEPTION;
      PRAGMA EXCEPTION_INIT(deadlock_detected, -60);
```

For further information on the subject see:

Technical Reference Appendix, pages 8 and 9

PL/SQL User's Guide and Reference Version 1.0, 800-20

REFERENCE ERROR REPORTING FUNCTIONS

SQLCODE and SQLERRM

- Provides information on the exception currently being handled
- Especially useful in the OTHERS handler

SQLCODE

- Returns the ORACLE error number of the exception, or 1 if it was a user-defined exception.

SQLERRM

- Returns the ORACLE error message associated with the current value of SQLCODE.
- Can also use any ORACLE error number as an argument.

Quick Notes

- If no exception is active...

  SQLCODE = 0

  SQLERRM = normal, successful completion
- SQLCODE and SQLERRM cannot be used within a SQL statement.

Reference Error Reporting Functions—cont'd

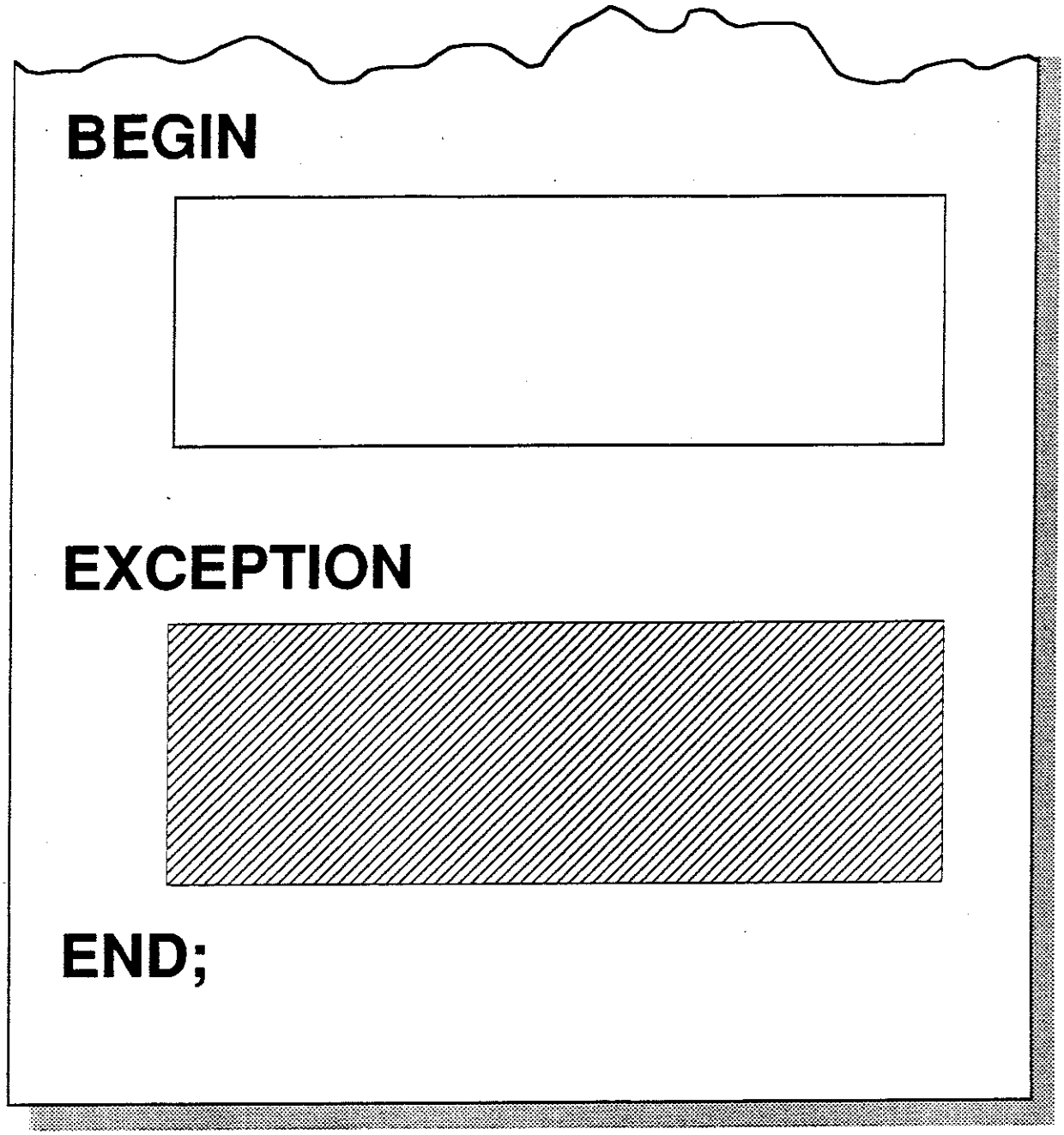
Example

```
DECLARE
    sqlcode_val    NUMBER;
    sqlerrm_val    CHAR(55);
BEGIN
...
EXCEPTION
    WHEN OTHERS THEN
        sqlcode_val := SQLCODE;
            -- Can't insert SQLCODE directly.

        sqlerrm_val := SUBSTR(SQLERRM,1,55);
            -- Can't insert SQLERRM directly.
            -- Also, SUBSTR should be used to ensure
            -- that returned string will fit into
            -- target variable.

        INSERT INTO temp (col1, message)
            VALUES (sqlcode_val, sqlerrm_val);
END;
```

HANDLE PL/SQL ERRORS: SUMMARY



Handle PL/SQL Errors: Summary—cont'd

Handle PL/SQL errors in the exception handler section.

- Predefined internal exceptions are raised automatically in response to an ORACLE error.
- User-defined exceptions must be declared and explicitly raised.
- When an exception is raised, the current block is searched for a handler. If there is no handler present, the exception propagates to the enclosing block.
- For ORACLE errors that do not correspond to a predefined internal exception, declare a user-defined exception and associate it with an ORACLE error using PRAGMA EXCEPTION_INIT.
- The functions SQLCODE and SQLERRM provide information on the exception currently being handled.

LAB 5-1

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

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—cont'd

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?

Lab 5-1—cont'd

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.