Review of PL/SQL

Block Structure for Anonymous PL/SQL Blocks

- `DECLARE` (optional)
  - Declare PL/SQL objects to be used within this block.
- `BEGIN` (mandatory)
  - Define the executable statements.
- `EXCEPTION` (optional)
  - Define the actions that take place if an error or exception arises.
- `END;` (mandatory)

Declaring PL/SQL Variables

- Syntax:

```
identifier [CONSTANT] datatype [NOT NULL]
      [:= | DEFAULT expr];
```

- Examples:

```
Declare
  v_hiredate      DATE;
  v_deptno        NUMBER(2) NOT NULL := 10;
  v_location      VARCHAR2(13) := 'Atlanta';
  c_ comm         CONSTANT NUMBER := 1400;
  v_count         BINARY_INTEGER := 0;
  v_valid         BOOLEAN NOT NULL := TRUE;
```

Declaring Variables with the %TYPE Attribute

Examples:

```
...
  v_ename                 employees.last_name%TYPE;
  v_balance               NUMBER(7,2);
  v_min_balance           v_balance%TYPE := 10;
...
```

Creating a PL/SQL Record

Declare variables to store the name, job, and salary of a new employee.

Example:

```
...
  TYPE emp_record_type IS RECORD
    (ename    VARCHAR2(25),
     job          VARCHAR2(10),
     sal          NUMBER(8,2));
  emp_record     emp_record_type;
...
```

%ROWTYPE Attribute

Examples:

- Declare a variable to store the same information about a department as is stored in the DEPARTMENTS table.

```
dept_record    departments%ROWTYPE;
```

- Declare a variable to store the same information about an employee as is stored in the EMPLOYEES table.

```
emp_record     employees%ROWTYPE;
```

Creating a PL/SQL Table

```
DECLARE
  TYPE ename_table_type IS TABLE OF
    employees.last_name%TYPE
    INDEX BY BINARY_INTEGER;
  TYPE hiredate_table_type IS TABLE OF DATE
    INDEX BY BINARY_INTEGER;
  ename_table     ename_table_type;
  hiredate_table  hiredate_table_type;
BEGIN
  ename_table(1) := 'CAMERON';
  hiredate_table(8) := SYSDATE + 7;
    IF ename_table.EXISTS(1) THEN
      INSERT INTO ...
    ...
END;
```

SELECT Statements in PL/SQL

The INTO clause is mandatory.

Example:

```
DECLARE
  v_deptid    NUMBER(4);
  v_loc       NUMBER(4);
BEGIN
  SELECT  department_id, location_id
  INTO    v_deptno, v_loc
  FROM    departments
  WHERE   department_name = 'Sales';
...
END;
```

Inserting Data

Add new employee information to the `EMPLOYEES` table.

Example:

```
DECLARE
  v_empid  employees.employee_id%TYPE;
BEGIN
  SELECT  employees_seq.NEXTVAL
  INTO    v_empno
  FROM    dual;
  INSERT INTO  employees(employee_id, last_name,
                         job_id, department_id)
  VALUES(v_empno, 'HARDING', 'PU_CLERK', 30);
END;
```

Updating Data

Increase the salary of all employees in the `EMPLOYEES` table who are purchasing clerks.

Example:

```
DECLARE
  v_sal_increase   employees.salary%TYPE := 2000;
BEGIN
  UPDATE  employees
  SET     salary = salary + v_sal_increase
  WHERE   job_id = 'PU_CLERK';
END;
```

Deleting Data

Delete rows that belong to department 190 from the `EMPLOYEES` table.

Example:

```
DECLARE
  v_deptid   employees.department_id%TYPE := 190;
BEGIN
  DELETE FROM employees
  WHERE department_id = v_deptid;
END;
```

COMMIT and ROLLBACK Statements

- Initiate a transaction with the first DML command to follow a COMMIT or ROLLBACK statement.
- Use COMMIT and ROLLBACK SQL statements to terminate a transaction explicitly.

SQL Cursor Attributes

Using SQL cursor attributes, you can test the outcome of your SQL statements.

SQL%ROWCOUNT	Number of rows affected by the most recent SQL statement (an integer value)
SQL%FOUND	Boolean attribute that evaluates to `TRUE` if the most recent SQL statement affects one or more rows
SQL%NOTFOUND	Boolean attribute that evaluates to `TRUE` if the most recent SQL statement does not affect any rows
SQL%ISOPEN	Boolean attribute that always evaluates to `FALSE` because PL/SQL closes implicit cursors immediately after they are executed

IF, THEN, and ELSIF Statements

For a given value entered, return a calculated value.

Example:

```
. . .
IF v_start > 100 THEN
  v_start := 2 * v_start;
ELSIF v_start >= 50 THEN
  v_start := 0.5 * v_start;
ELSE
  v_start := 0.1 * v_start;
END IF;
. . .
```

Basic Loop

Example:

```
DECLARE
  v_ordid     order_items.order_id%TYPE := 101;
  v_counter   NUMBER(2) := 1;
BEGIN
  LOOP
    INSERT INTO order_items(order_id,line_item_id)
    VALUES(v_ordid, v_counter);
    v_counter := v_counter + 1;
    EXIT WHEN v_counter > 10;
  END LOOP;
END;
```

FOR Loop

Insert the first 10 new line items for order number 101.

Example:

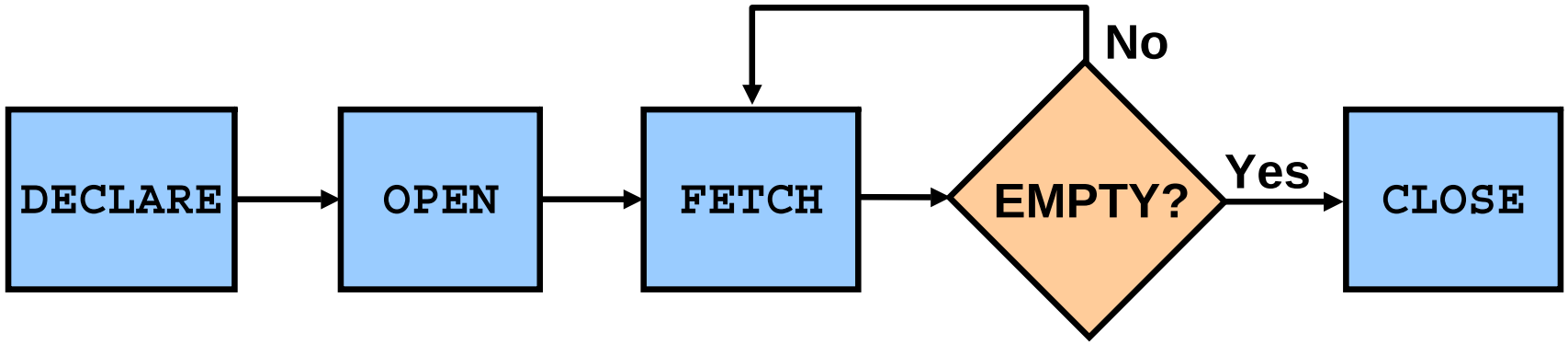
```
DECLARE
  v_ordid    order_items.order_id%TYPE := 101;
BEGIN
  FOR i IN 1..10 LOOP
    INSERT INTO order_items(order_id,line_item_id)
    VALUES(v_ordid, i);
  END LOOP;
END;
```

WHILE Loop

Example:

```
ACCEPT p_price PROMPT 'Enter the price of the item: '
ACCEPT p_itemtot -
 PROMPT 'Enter the maximum total for purchase of item: '
DECLARE
...
v_qty                   NUMBER(8) := 1;
v_running_total         NUMBER(7,2) := 0;
BEGIN
  ...
  WHILE v_running_total < &p_itemtot LOOP
    ...
  v_qty := v_qty + 1;
  v_running_total := v_qty * &p_price;
  END LOOP;
...
```

Controlling Explicit Cursors



- Create a named SQL area.
- Identify the active set.
- Load the current row into variables.
- Test for existing rows.
- Return to *FETCH* if rows are found.
- Release the active set.

Declaring the Cursor

Example:

```
DECLARE
  CURSOR c1 IS
    SELECT employee_id, last_name
    FROM   employees;

  CURSOR c2 IS
    SELECT *
    FROM   departments
    WHERE  department_id = 10;
BEGIN
  ...
```

Opening the Cursor

Syntax:

```
OPEN cursor_name;
```

- Open the cursor to execute the query and identify the active set.
- If the query returns no rows, no exception is raised.
- Use cursor attributes to test the outcome after a fetch.

Fetching Data from the Cursor

Examples:

```
FETCH c1 INTO v_empid, v_ename;
```

```
...
OPEN defined_cursor;
LOOP
  FETCH defined_cursor INTO defined_variables
  EXIT WHEN ...;
  ...
    -- Process the retrieved data
  ...
END;
```

Closing the Cursor

Syntax:

```
CLOSE    cursor_name;
```

- Close the cursor after completing the processing of the rows.
- Reopen the cursor, if required.
- Do not attempt to fetch data from a cursor after it has been closed.

Explicit Cursor Attributes

Obtain status information about a cursor.

Attribute	Type	Description
%ISOPEN	BOOLEAN	Evaluates to TRUE if the cursor is open
%NOTFOUND	BOOLEAN	Evaluates to TRUE if the most recent fetch does not return a row
%FOUND	BOOLEAN	Evaluates to TRUE if the most recent fetch returns a row; complement of %NOTFOUND
%ROWCOUNT	NUMBER	Evaluates to the total number of rows returned so far

Cursor FOR Loops

Retrieve employees one by one until there are no more left.

Example:

```
DECLARE
  CURSOR c1 IS
    SELECT employee_id, last_name
    FROM   employees;
BEGIN
  FOR emp_record IN c1 LOOP
         -- implicit open and implicit fetch occur
    IF emp_record.employee_id = 134 THEN
      ...
  END LOOP; -- implicit close occurs
END;
```

FOR UPDATE Clause

Retrieve the orders for amounts over $1,000 that were processed today.

Example:

```
DECLARE
  CURSOR c1 IS
    SELECT customer_id, order_id
    FROM   orders
    WHERE  order_date = SYSDATE
      AND  order_total > 1000.00
    ORDER BY customer_id
    FOR UPDATE NOWAIT;
```

WHERE CURRENT OF Clause

Example:

```
DECLARE
  CURSOR c1 IS
    SELECT salary FROM employees
    FOR UPDATE OF salary NOWAIT;
BEGIN
  ...
  FOR emp_record IN c1 LOOP
    UPDATE ...
      WHERE CURRENT OF c1;
    ...
  END LOOP;
  COMMIT;
END;
```

Trapping Predefined Oracle Server Errors

- Reference the standard name in the exception-handling routine.
- Sample predefined exceptions:
  - `NO_DATA_FOUND`
  - `TOO_MANY_ROWS`
  - `INVALID_CURSOR`
  - `ZERO_DIVIDE`
  - `DUP_VAL_ON_INDEX`

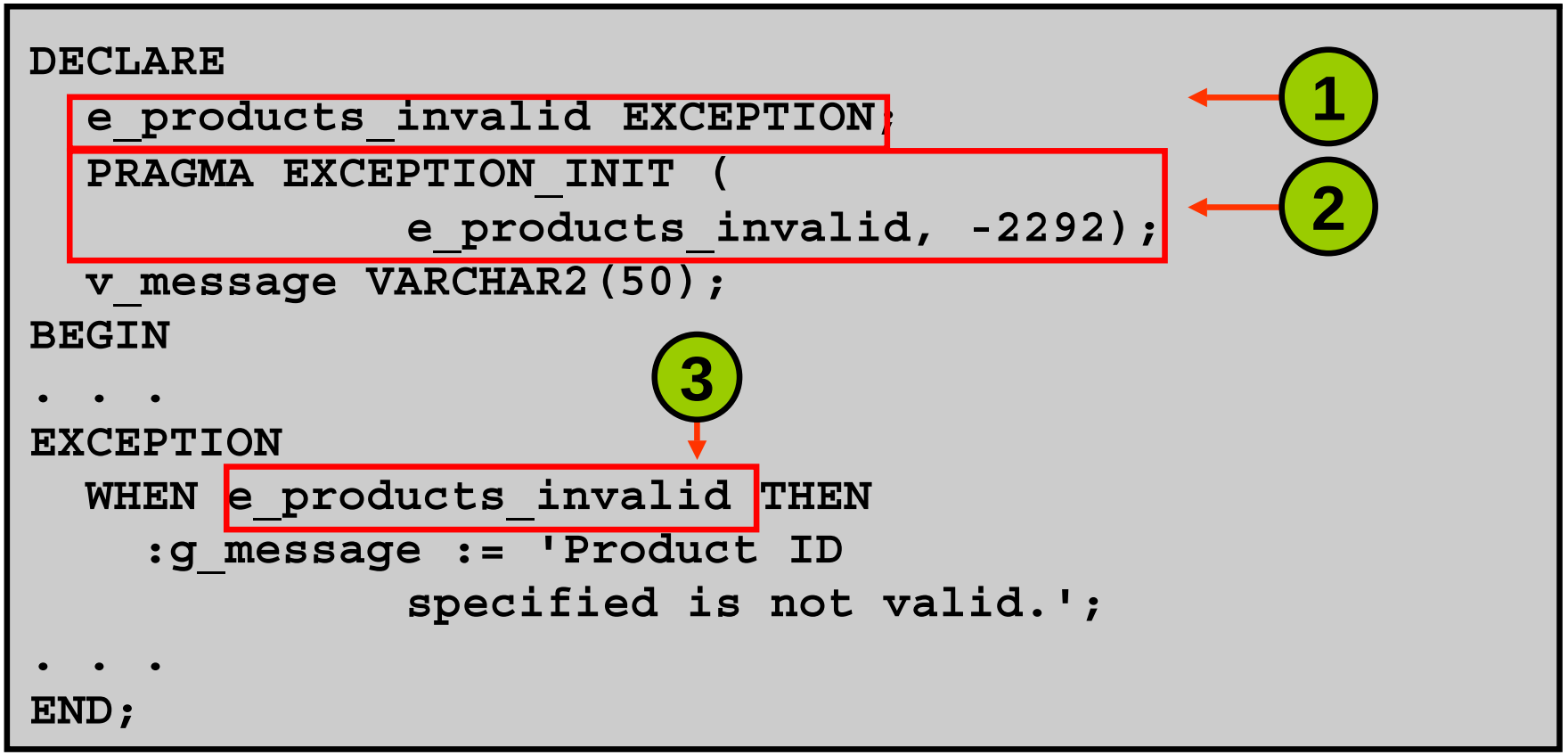
Trapping Predefined Oracle Server Errors: Example

Syntax:

```
BEGIN  SELECT ... COMMIT;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    statement1;
    statement2;
  WHEN TOO_MANY_ROWS THEN
    statement1;
  WHEN OTHERS THEN
    statement1;
    statement2;
    statement3;
END;
```

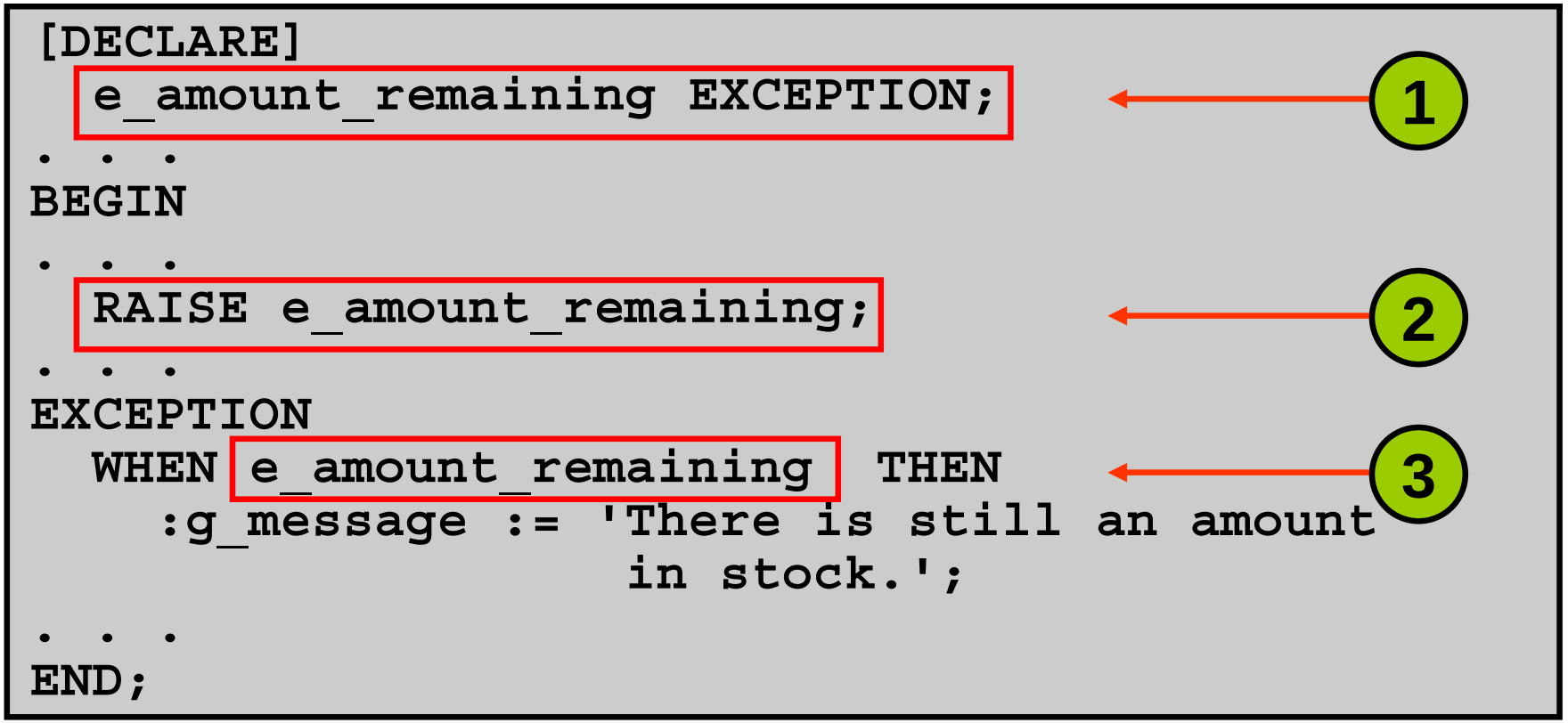
Non-Predefined Error

Trap for Oracle server error number –2292, which is an integrity constraint violation.



User-Defined Exceptions

Example:



RAISE_APPLICATION_ERROR Procedure

Syntax:

```
raise_application_error (error_number,
                         message[, {TRUE | FALSE}]);
```

- Enables you to issue user-defined error messages from stored subprograms
- Is called from an executing stored subprogram only

`RAISE_APPLICATION_ERROR` Procedure

- Is used in two different places:
  - `Executable` section
  - `Exception` section
- Returns error conditions to the user in a manner consistent with other Oracle server errors