P
L
/
S
Q
L

Oracle11*g*:
PL/SQL Programming

Chapter 4

Cursors and Exception Handling

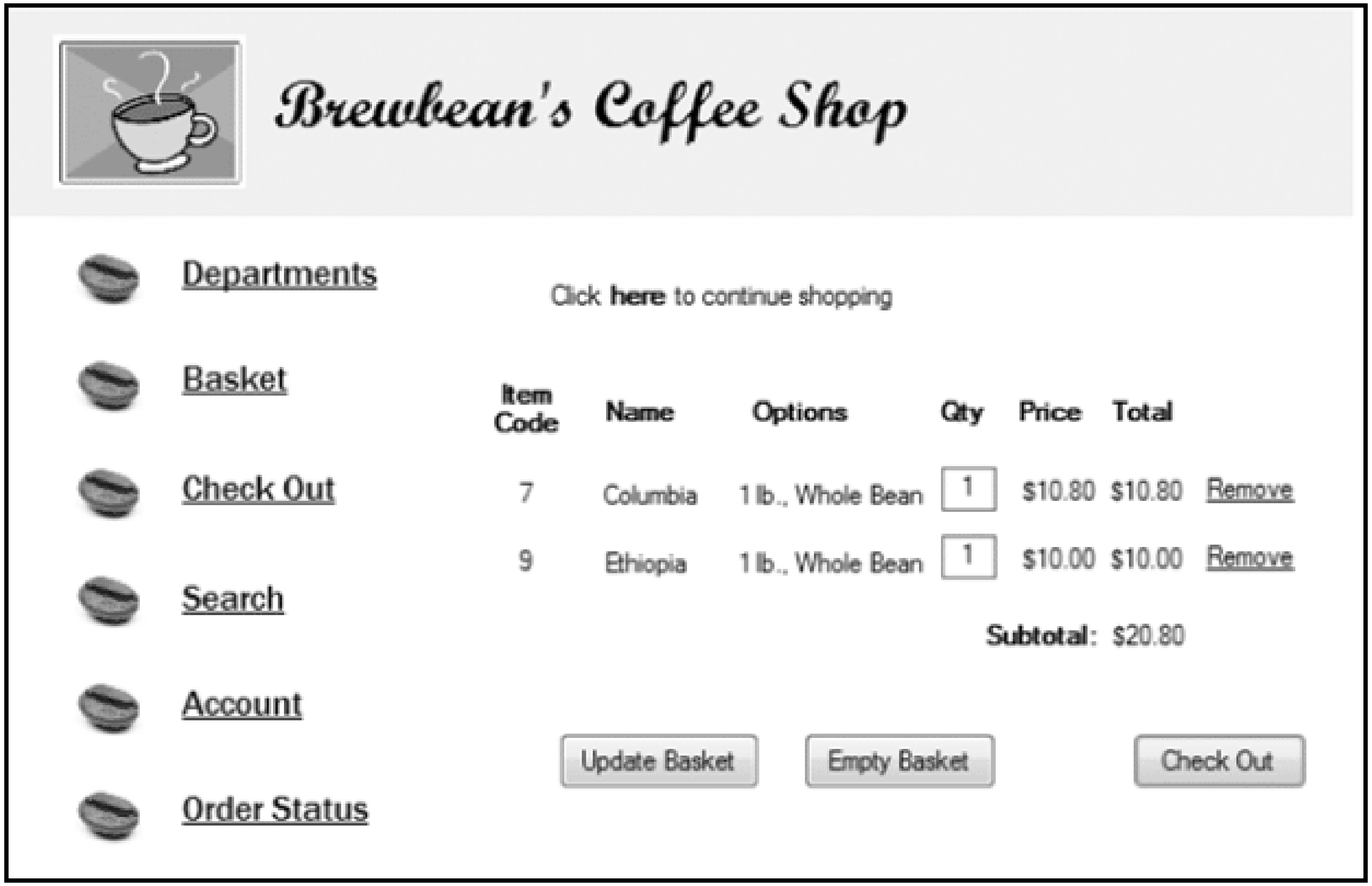
Chapter Objectives

P
L
/
S
Q
L

- After completing this lesson, you should be able to understand:
  - Manipulating data with cursors
  - Using bulk-processing features
  - Managing errors with exception handlers
  - Addressing exception-handling issues, such as RAISE_APPLICATION_ERROR and propagation
  - Documenting code with comments

Brewbean's Challenge

- Processing multiple data rows



Click **here** to continue shopping

Item Code	Name	Options	Qty	Price	Total	
7	Columbia	1 lb., Whole Bean	1	$10.80	$10.80	Remove
9	Ethiopia	1 lb., Whole Bean	1	$10.00	$10.00	Remove
				Subtotal:	$20.80	

Cursors

P
L
/
S
Q
L

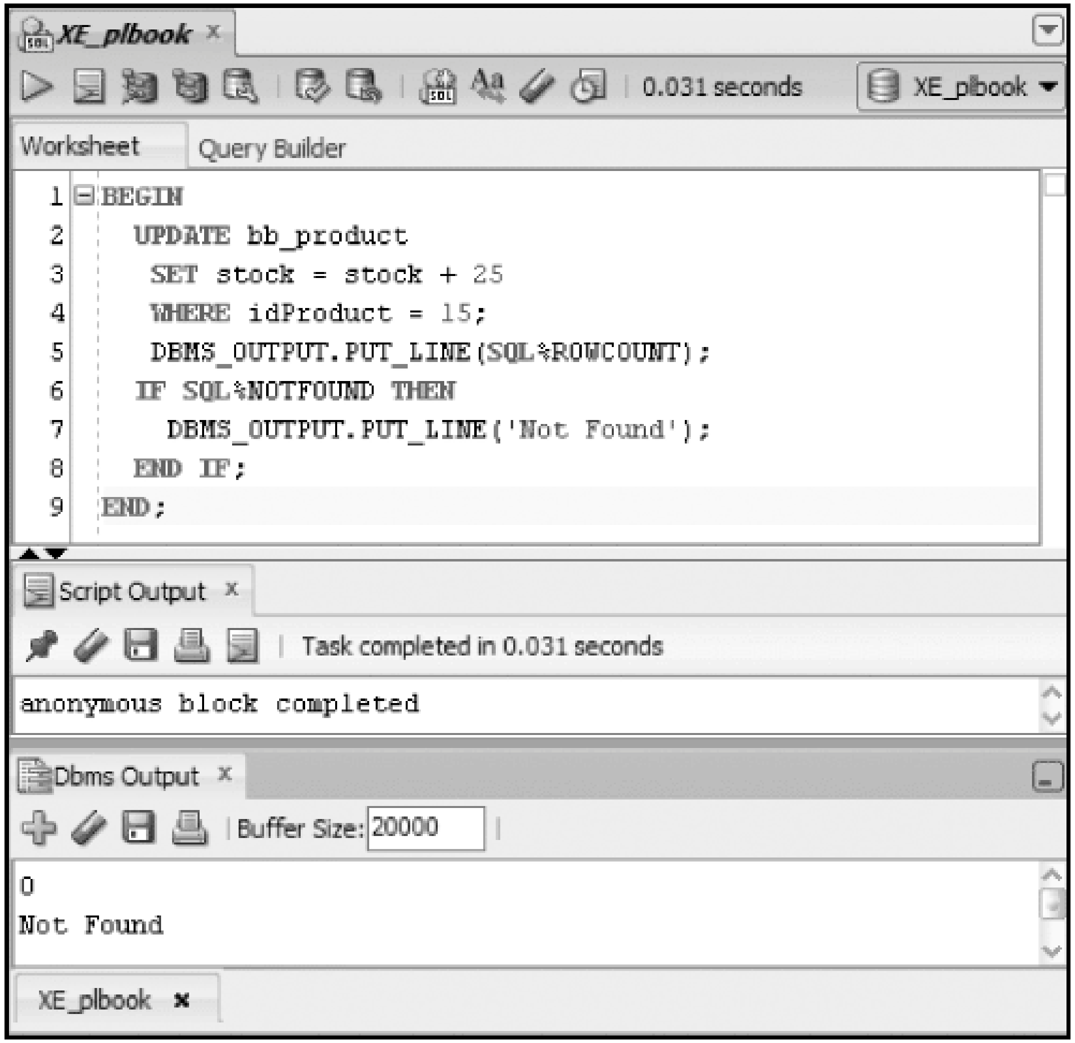
- Work area in which SQL statement is processed
- Implicit cursor – declared automatically for DML and SELECT statements
- Explicit cursor – declared and managed programmatically to handle a set of rows returned by a SELECT statement
- Cursor variable – reference or pointer to a work area or cursor

Cursor Attributes

Attribute Name	Data type	Description
%ROWCOUNT	Number	Number of rows affected by the SQL statement
%FOUND	Boolean	TRUE if at least one row is affected by the SQL statement, otherwise FALSE
%NOTFOUND	Boolean	TRUE if no rows are affected by the SQL statement, otherwise FALSE

Implicit Cursor

P
L
/
S
Q
L



```
BEGIN
  UPDATE bb_product
   SET stock = stock + 25
   WHERE idProduct = 15;
   DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT);
  IF SQL%NOTFOUND THEN
    DBMS_OUTPUT.PUT_LINE('Not Found');
  END IF;
END;
```

```
anonymous block completed
```

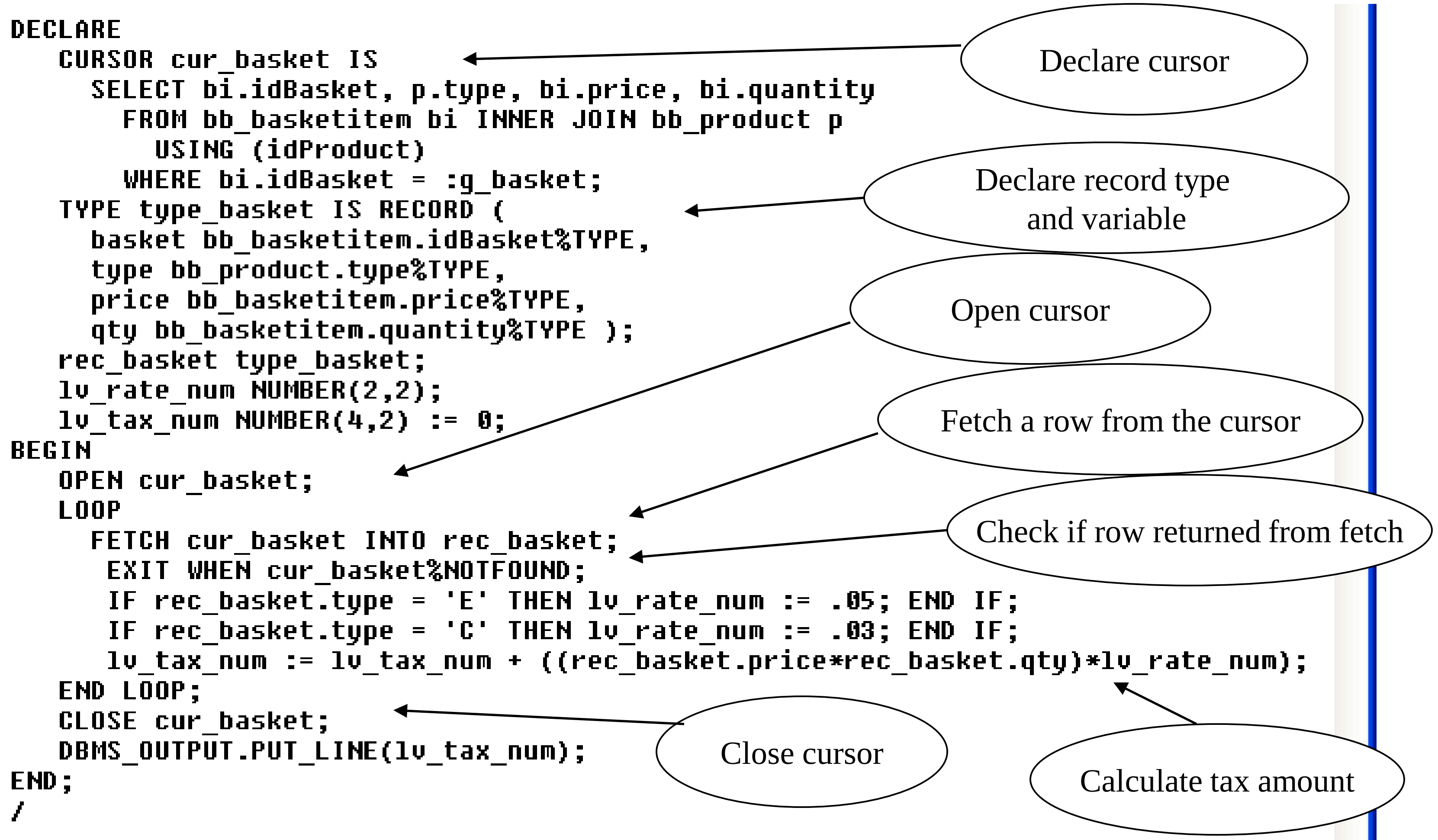
```
0
Not Found
```

P
L
/
S
Q
L

Explicit Cursor

Step	Step Activity	Activity Description
1	DECLARE	Creates a named cursor identified by a SELECT statement. The SELECT statement does not include an INTO clause. Values in the cursor are moved to PL/SQL variables with the FETCH step.
2	OPEN	Processes the query and creates the active set of rows available in the cursor.
3	FETCH	Retrieves a row from the cursor into block variables. Each consecutive FETCH issued will retrieve the next row in the cursor until all rows have been retrieved.
4	CLOSE	Clears the active set of rows and frees the memory area used for the cursor.

Explicit Cursor Example



```
DECLARE
   CURSOR cur_basket IS
     SELECT bi.idBasket, p.type, bi.price, bi.quantity
       FROM bb_basketitem bi INNER JOIN bb_product p
         USING (idProduct)
       WHERE bi.idBasket = :g_basket;
   TYPE type_basket IS RECORD (
     basket bb_basketitem.idBasket%TYPE,
     type bb_product.type%TYPE,
     price bb_basketitem.price%TYPE,
     qty bb_basketitem.quantity%TYPE );
   rec_basket type_basket;
   lv_rate_num NUMBER(2,2);
   lv_tax_num NUMBER(4,2) := 0;
BEGIN
   OPEN cur_basket;
   LOOP
     FETCH cur_basket INTO rec_basket;
      EXIT WHEN cur_basket%NOTFOUND;
      IF rec_basket.type = 'E' THEN lv_rate_num := .05; END IF;
      IF rec_basket.type = 'C' THEN lv_rate_num := .03; END IF;
      lv_tax_num := lv_tax_num + ((rec_basket.price*rec_basket.qty)*lv_rate_num);
   END LOOP;
   CLOSE cur_basket;
   DBMS_OUTPUT.PUT_LINE(lv_tax_num);
END;
/
```

Declare cursor
Declare record type and variable
Open cursor
Fetch a row from the cursor
Check if row returned from fetch
Close cursor
Calculate tax amount

Cursor FOR Loop

- Handles tasks automatically for processing each row returned by a cursor (record declaration, fetch, ending loop)
- Use FOR UPDATE and WHERE CURRENT OF clauses for record locking

P
L
/
S
Q
L

Cursor FOR Loop Example

```
DECLARE
   CURSOR cur_prod IS
        SELECT type, price
          FROM bb_product
          WHERE active = 1
        FOR UPDATE NOWAIT;
   lv_sale bb_product.saleprice%TYPE;
BEGIN
  FOR rec_prod IN cur_prod LOOP
     IF rec_prod.type = 'C' THEN lv_sale := rec_prod.price * .9;
      ELSIF rec_prod.type = 'E' THEN lv_sale := rec_prod.price * .95;
     END IF;
     UPDATE bb_product
       SET saleprice = lv_sale
       WHERE CURRENT OF cur_prod;
   END LOOP;
COMMIT;
END;
```

Cursors with Parameters

- Use parameters to make dynamic
- Parameters are values passed to the cursor when it is opened
- Enables the cursor to retrieve different data based on the input values

Cursors with Parameters

P
L
/
S
Q
L

```
DECLARE
    CURSOR cur_order (p_basket NUMBER) IS
       SELECT idBasket, idProduct, price, quantity
        FROM bb_basketitem
        WHERE idBasket = p_basket;
    lv_bask1_num bb_basket.idbasket%TYPE := 6;
    lv_bask2_num bb_basket.idbasket%TYPE := 10;
BEGIN
   FOR rec_order IN cur_order(lv_bask1_num) LOOP
       DBMS_OUTPUT.PUT_LINE(rec_order.idBasket || ' - ' ||
                            rec_order.idProduct || ' - ' || rec_order.price);
   END LOOP;
   FOR rec_order IN cur_order(lv_bask2_num) LOOP
        DBMS_OUTPUT.PUT_LINE(rec_order.idBasket || ' - ' ||
                            rec_order.idProduct || ' - ' || rec_order.price);
   END LOOP;
END;
```

P
L
/
S
Q
L

Cursor Variable

- More efficiently handles data returned by query by returning a pointer to the work area rather than the actual result set
- The same cursor variable can be used for different query statements

Cursor Variable Example

P
L
/
S
Q
L

```
DECLARE
     cv_prod SYS_REFCURSOR;
     rec_item bb_basketitem%ROWTYPE;
     rec_status bb_basketstatus%ROWTYPE;
     lv_input1_num NUMBER(2) := 2;
     lv_input2_num NUMBER(2) := 3;
BEGIN
     IF lv_input1_num = 1 THEN
         OPEN cv_prod FOR SELECT * FROM bb_basketitem
           WHERE idBasket = lv_input2_num;
          LOOP
            FETCH cv_prod INTO rec_item;
            EXIT WHEN cv_prod%NOTFOUND;
            DBMS_OUTPUT.PUT_LINE(rec_item.idProduct);
          END LOOP;
```

Example (continued)

P
L
/
S
Q
L

```
  ELSIF lv_input1_num = 2 THEN
     OPEN cv_prod FOR SELECT * FROM bb_basketstatus
                                       WHERE idBasket = lv_input2_num;
        LOOP
           FETCH cv_prod INTO rec_status;
           EXIT WHEN cv_prod%NOTFOUND;
           DBMS_OUTPUT.PUT_LINE(rec_status.idStage || ' - '
                                                  || rec_status.dtstage);
        END LOOP;
  END IF;
END;
```

Bulk-processing

- Improve performance of multirow queries and DML statements
- Processes groups of rows without context switching between the SQL and PL/SQL processing engine
- Use in FETCH with LIMIT clause
- FORALL option with DML activity

Bulk-processing (Query)

P
L
/
S
Q
L

```
DECLARE
      CURSOR cur_item IS
            SELECT *
            FROM bb_basketitem;
      TYPE type_item IS TABLE OF cur_item%ROWTYPE
                              INDEX BY PLS_INTEGER;
       tbl_item type_item;
BEGIN
       OPEN cur_item;
       LOOP
            FETCH cur_item BULK COLLECT INTO tbl_item LIMIT 1000;
            FOR i IN 1..tbl_item.COUNT LOOP
               DBMS_OUTPUT.PUT_LINE(tbl_item(i).idBasketitem || ' -'
                                                                   || tbl_item(i).idProduct);
            END LOOP;
            EXIT WHEN cur_item%NOTFOUND;
       END LOOP;
       CLOSE cur_item;
END;
```

Bulk-processing (DML)

```
DECLARE
    TYPE emp_type IS TABLE OF NUMBER INDEX
           BY BINARY_INTEGER;
    emp_tbl emp_type;
BEGIN
     SELECT empID
       BULK COLLECT INTO emp_tbl
       FROM employees
         WHERE classtype = '100';
     FORALL i IN d_emp_tbl.FIRST .. emp_tbl.LAST
       UPDATE employees
           SET raise = salary * .06
           WHERE empID = emp_tbl(i);
       COMMIT;
END;
```

Exception Handlers

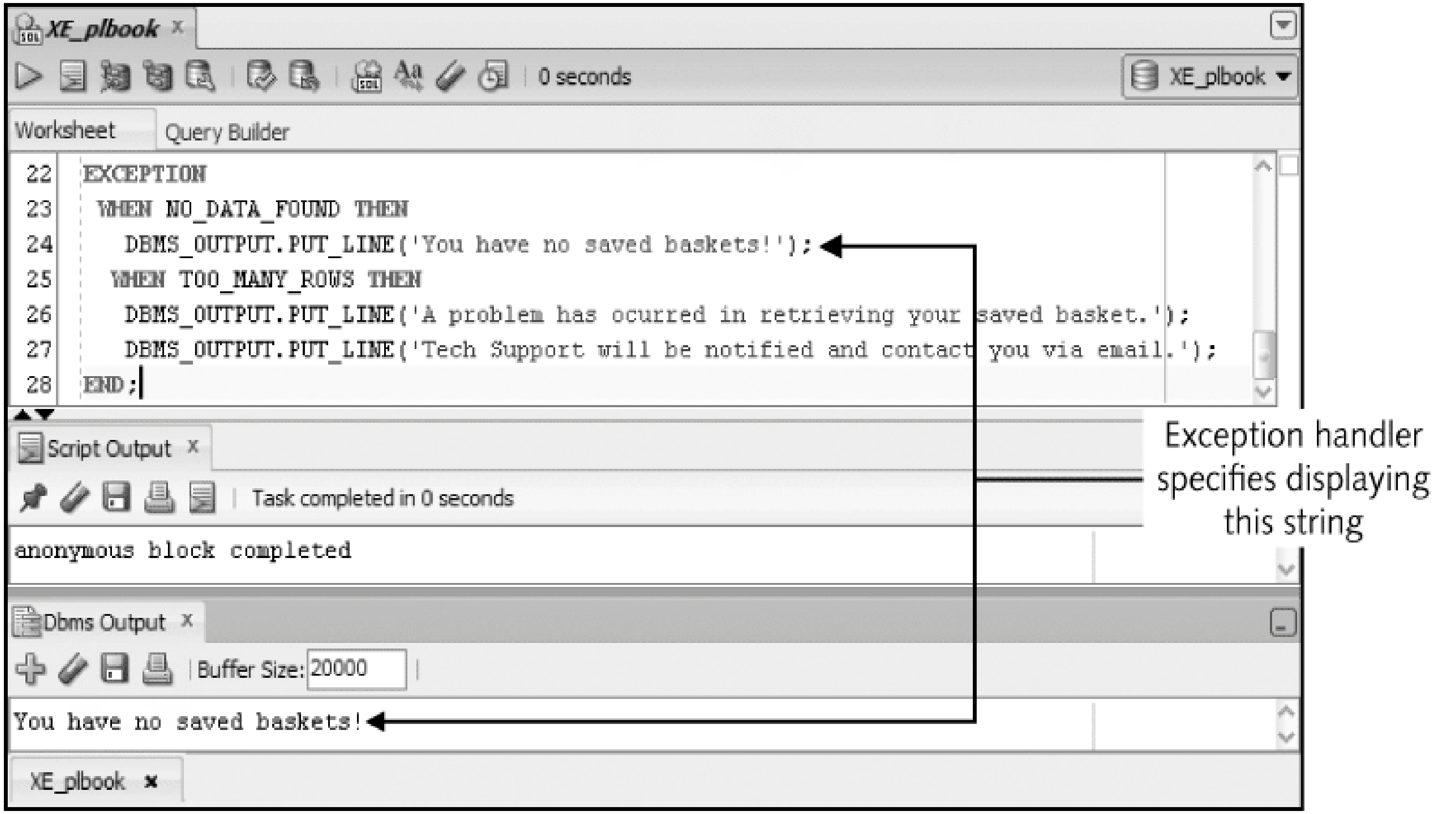
- Used to capture error conditions and handle the processing to allow the application to continue
- Placed in the EXCEPTION section of a PL/SQL block
- Two types of errors
  1. Oracle errors (Predefined and Non-Predefined)
  2. User-defined errors
- RAISE_APPLICATION_ERROR

Predefined Oracle Errors

P
L
/
S
Q
L

Exception Name	Description
NO_DATA_FOUND	A SELECT statement in a PL/SQL block retrieves no rows or a nonexistent row of an index-by table is referenced
TOO_MANY_ROWS	A SELECT statement in a PL/SQL block retrieves more than one row
CASE_NOT_FOUND	No WHEN clause in the CASE statement is processed
ZERO_DIVIDE	Attempted division by zero
DUP_VAL_ON_INDEX	Attempted violation of a unique or primary key column constraint

Predefined Error Example



```
EXCEPTION
 WHEN NO_DATA_FOUND THEN
   DBMS_OUTPUT.PUT_LINE('You have no saved baskets!');
  WHEN TOO_MANY_ROWS THEN
   DBMS_OUTPUT.PUT_LINE('A problem has ocurred in retrieving your saved basket.');
   DBMS_OUTPUT.PUT_LINE('Tech Support will be notified and contact you via email.');
END;
```

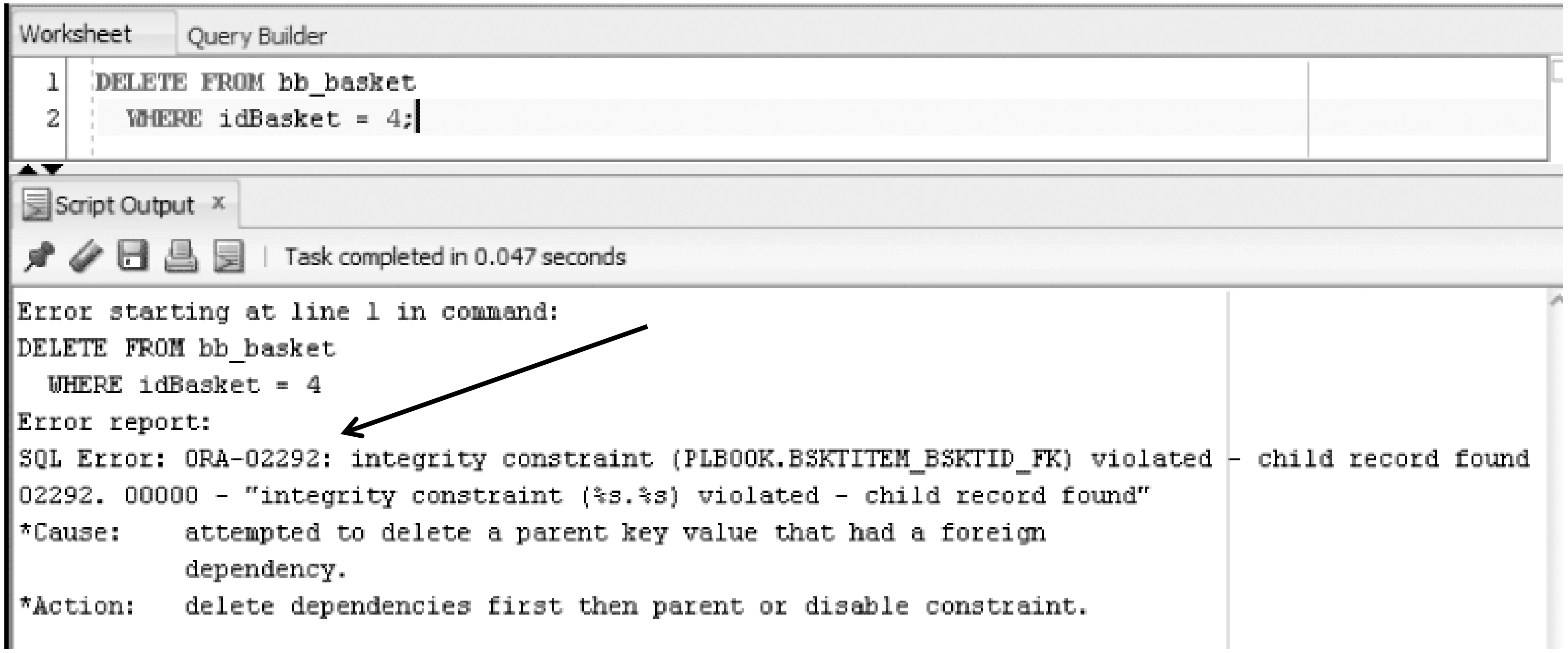
```
anonymous block completed
```

```
You have no saved baskets!
```

Undefined Error

P
L
/
S
Q
L

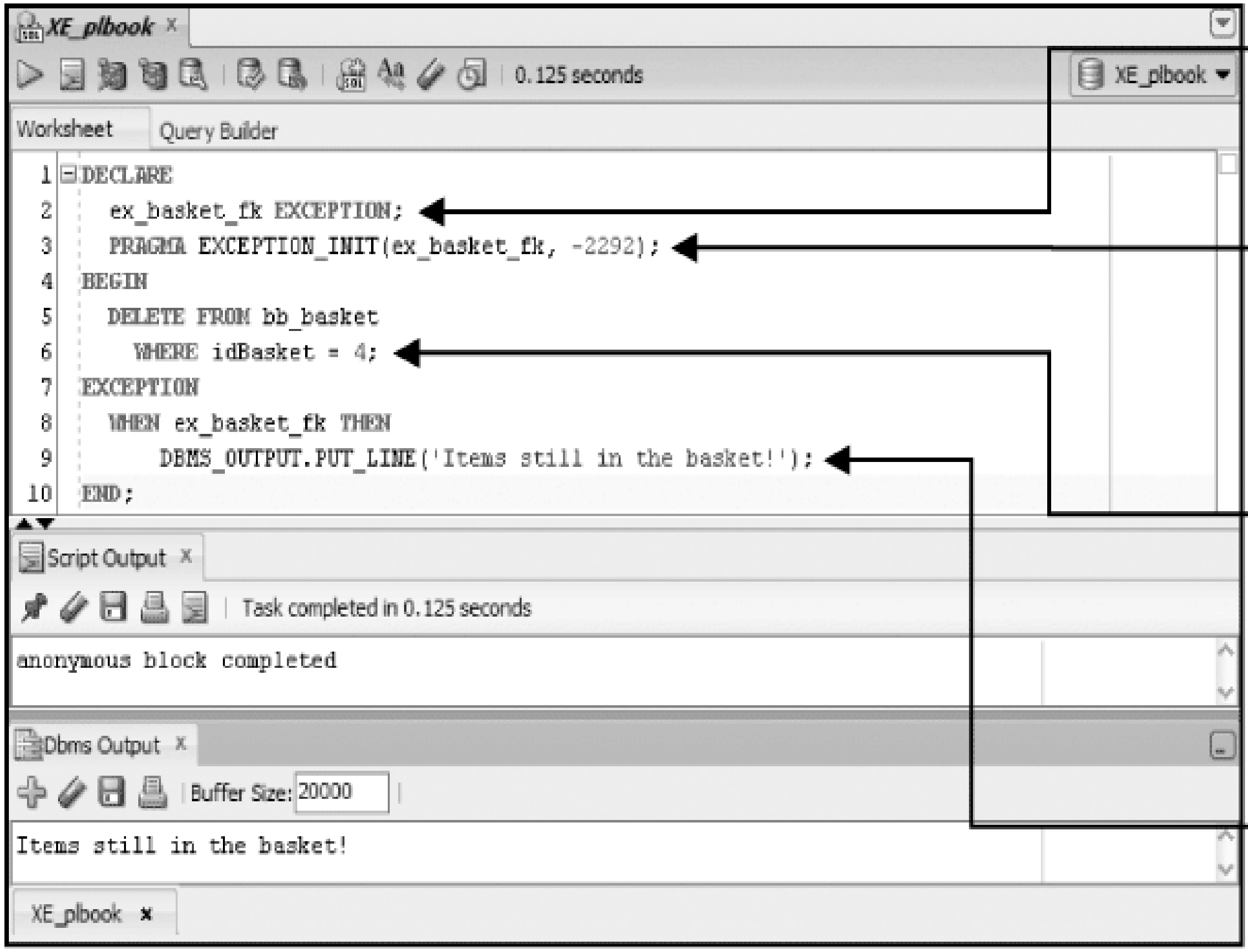
- Identify possible errors for statements in a block



```
DELETE FROM bb_basket
  WHERE idBasket = 4;
```

```
Error starting at line 1 in command:
DELETE FROM bb_basket
  WHERE idBasket = 4
Error report:
SQL Error: ORA-02292: integrity constraint (PLBOOK.BSKTITEM_BSKTID_FK) violated - child record found
02292. 00000 - "integrity constraint (%s.%s) violated - child record found"
*Cause:    attempted to delete a parent key value that had a foreign
           dependency.
*Action:   delete dependencies first then parent or disable constraint.
```

Handler Added



```
DECLARE
  ex_basket_fk EXCEPTION;
  PRAGMA EXCEPTION_INIT(ex_basket_fk, -2292);
BEGIN
  DELETE FROM bb_basket
    WHERE idBasket = 4;
EXCEPTION
  WHEN ex_basket_fk THEN
      DBMS_OUTPUT.PUT_LINE('Items still in the basket!');
END;
```

```
anonymous block completed
```

```
Items still in the basket!
```

Declare an exception name

Associate an Oracle error number with the exception name

Foreign key error occurs because of existing rows in the BB_BASKETITEM table

Exception handler runs if the `DELETE` statement raises foreign key error -2292

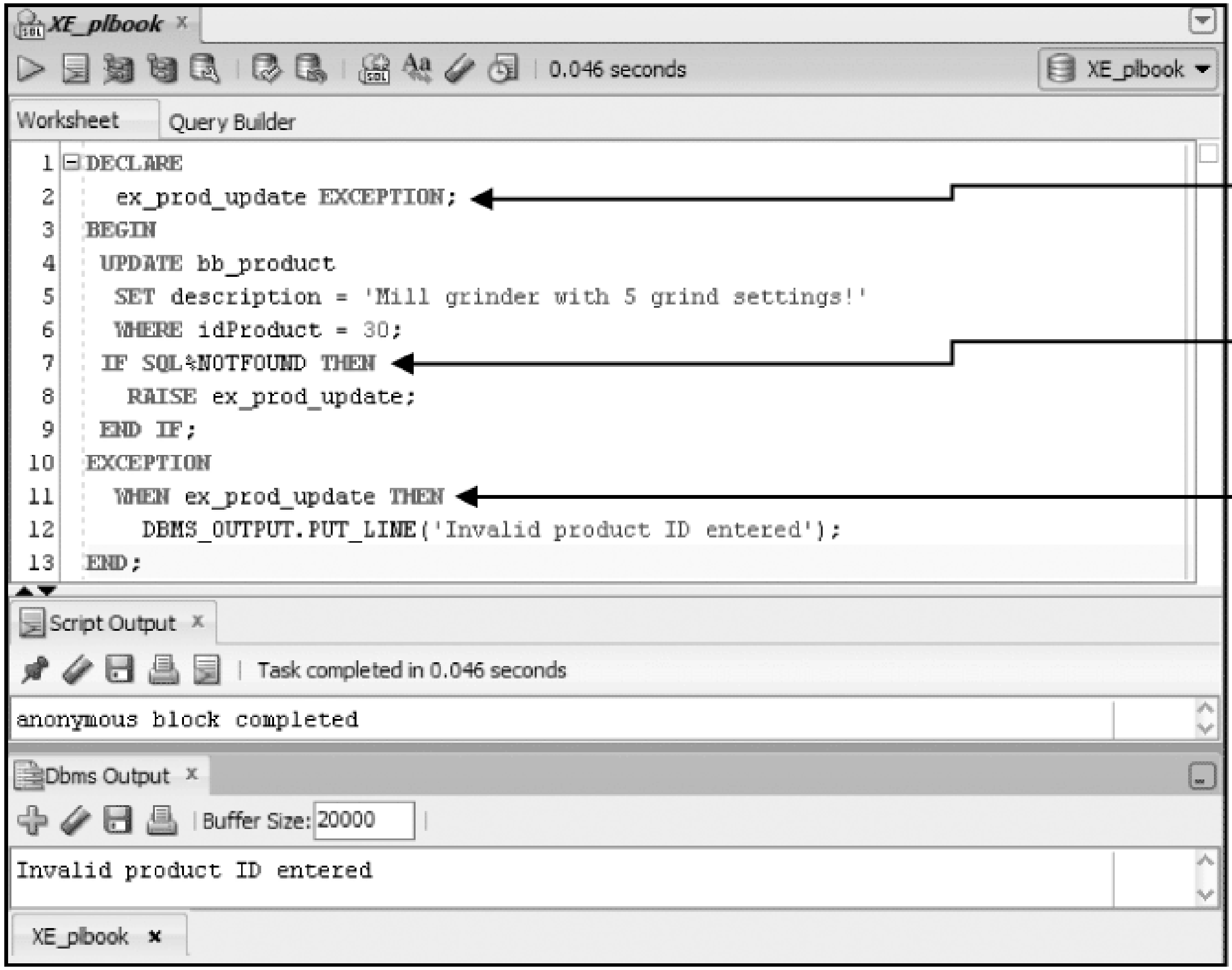
User-Defined Exception

P
L
/
S
Q
L

- No system error is raised
- Raise errors to enforce business rules
- Once error is raised, the remaining statements in the executable sections are not executed
- Processing moves to the exception area of the block

P
L
/
S
Q
L

User-Defined Exception Example



```
DECLARE
  ex_prod_update EXCEPTION;
BEGIN
 UPDATE bb_product
  SET description = 'Mill grinder with 5 grind settings!'
  WHERE idProduct = 30;
 IF SQL%NOTFOUND THEN
   RAISE ex_prod_update;
 END IF;
EXCEPTION
  WHEN ex_prod_update THEN
    DBMS_OUTPUT.PUT_LINE('Invalid product ID entered');
END;
```

Task completed in 0.046 seconds

```
anonymous block completed
```

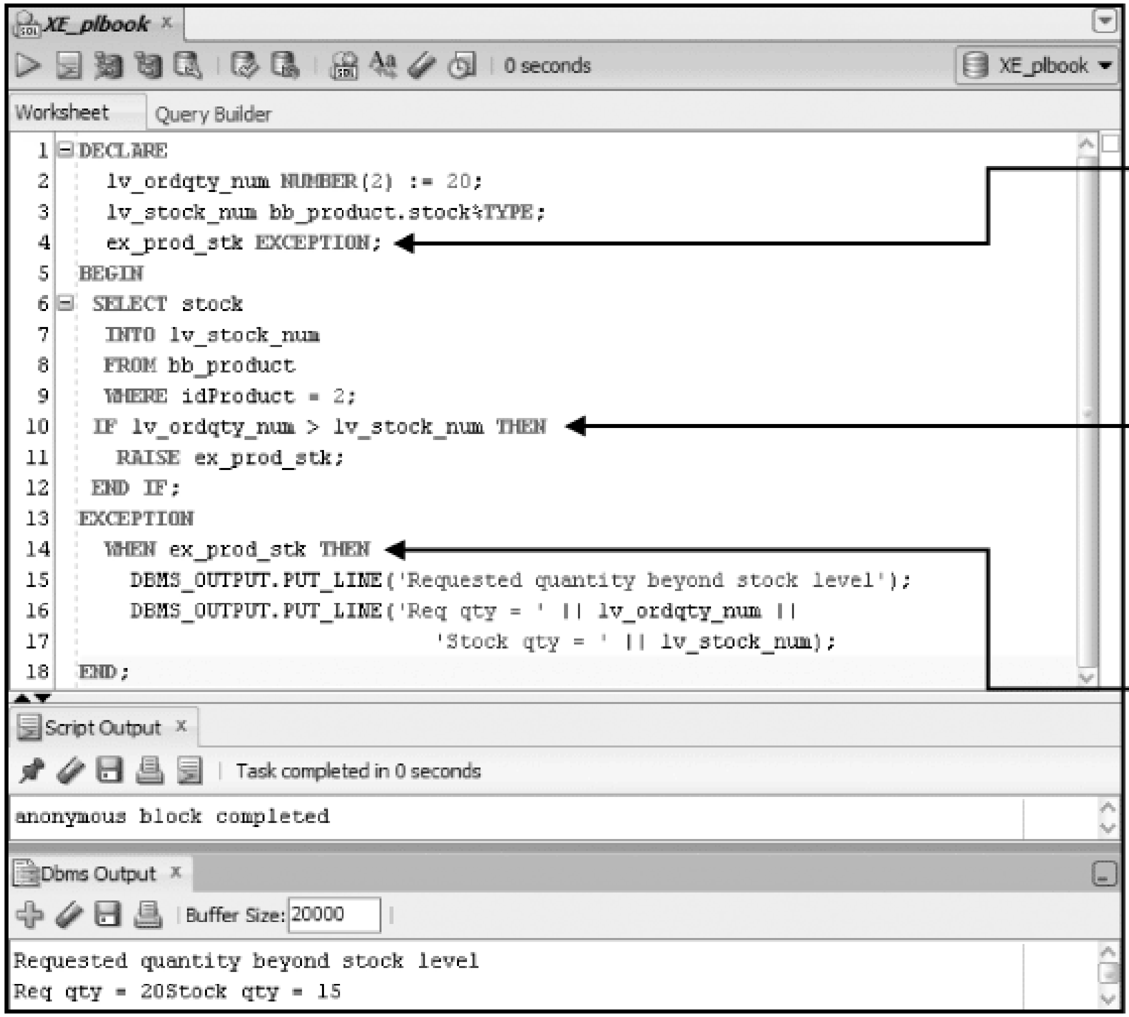
```
Invalid product ID entered
```

Declare an exception name

If no rows are updated, raise the exception

Establish an exception handler

User-Defined Exception Example



```
DECLARE
  lv_ordqty_num NUMBER(2) := 20;
  lv_stock_num bb_product.stock%TYPE;
  ex_prod_stk EXCEPTION;
BEGIN
 SELECT stock
  INTO lv_stock_num
  FROM bb_product
  WHERE idProduct = 2;
 IF lv_ordqty_num > lv_stock_num THEN
   RAISE ex_prod_stk;
 END IF;
EXCEPTION
  WHEN ex_prod_stk THEN
    DBMS_OUTPUT.PUT_LINE('Requested quantity beyond stock level');
    DBMS_OUTPUT.PUT_LINE('Req qty = ' || lv_ordqty_num ||
                         'Stock qty = ' || lv_stock_num);
END;
```

```
anonymous block completed
```

```
Requested quantity beyond stock level
Req qty = 20Stock qty = 15
```

Declare an exception name

If the quantity requested is greater than the quantity in stock, raise the exception

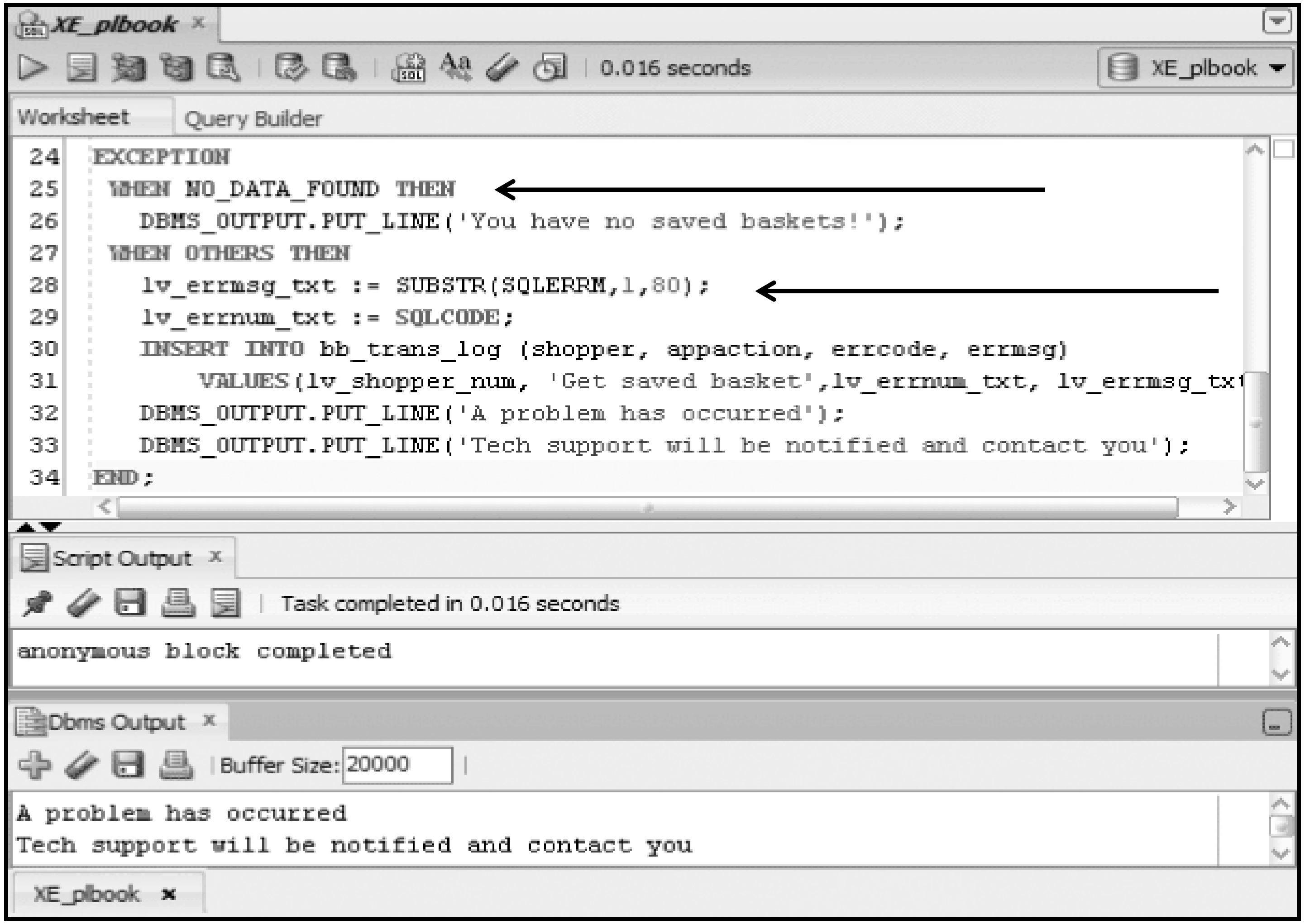
Establish an exception handler

Additional Exception Concepts

P
L
/
S
Q
L

- WHEN OTHERS – traps all errors not specifically addressed by an exception handler and used for handling unanticipated errors
- SQLCODE and SQLERRM – functions used to identify the error code and message, especially in application, testing to identify unanticipated errors

Example



```
EXCEPTION
 WHEN NO_DATA_FOUND THEN
   DBMS_OUTPUT.PUT_LINE('You have no saved baskets!');
 WHEN OTHERS THEN
   lv_errmsg_txt := SUBSTR(SQLERRM,1,80);
   lv_errnum_txt := SQLCODE;
   INSERT INTO bb_trans_log (shopper, appaction, errcode, errmsg)
       VALUES(lv_shopper_num, 'Get saved basket',lv_errnum_txt, lv_errmsg_tx
   DBMS_OUTPUT.PUT_LINE('A problem has occurred');
   DBMS_OUTPUT.PUT_LINE('Tech support will be notified and contact you');
END;
```

```
anonymous block completed
```

```
A problem has occurred
Tech support will be notified and contact you
```

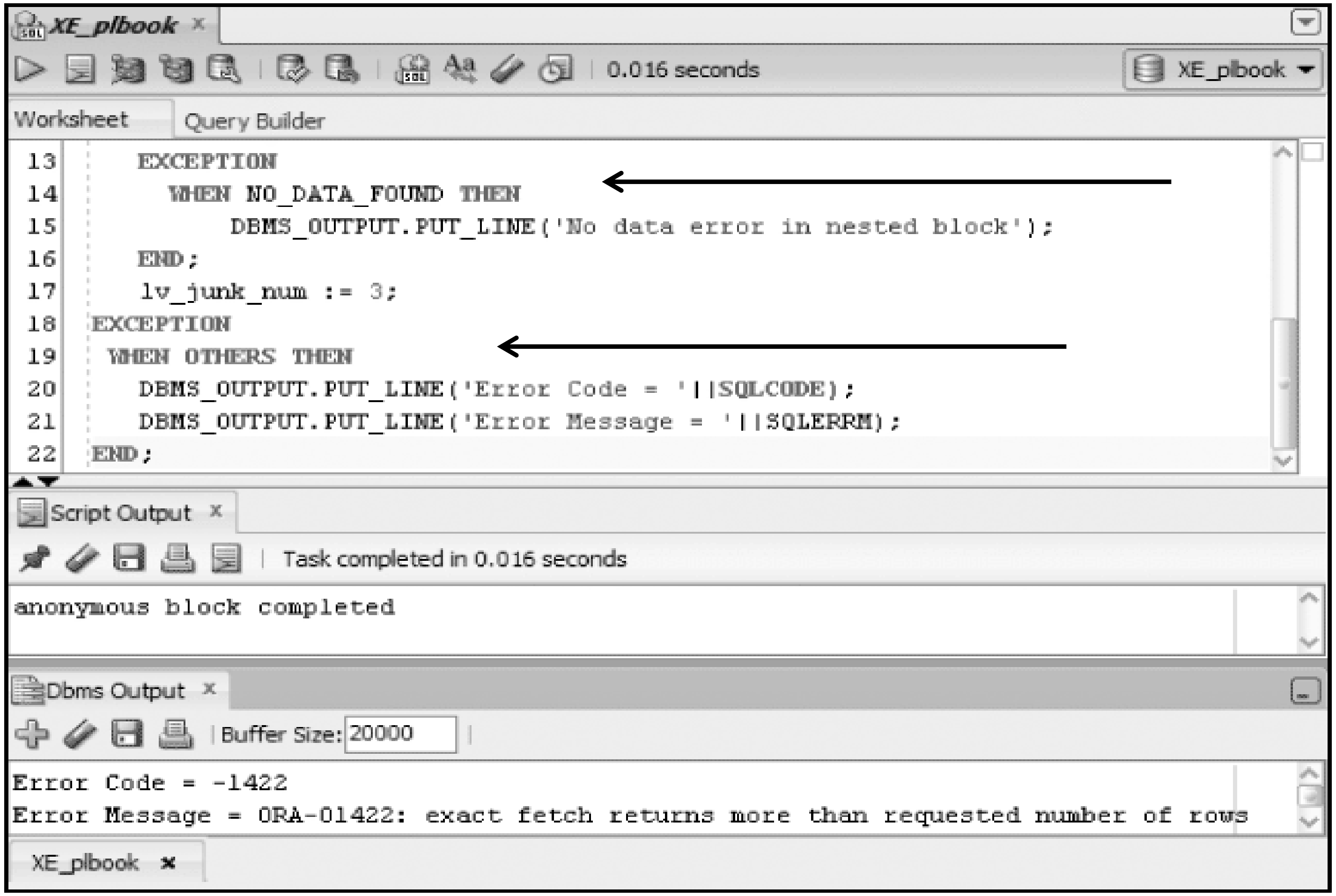
Exception Propagation

P
L
/
S
Q
L

- Exception handling in nested blocks
- Exception raised in a block will first look for handler in the exception section of that block, if no handler found, execution will move to the exception section of the enclosing block
- Error in DECLARE section propagates directly to exception section of the enclosing block
- Error in exception handler propagates to exception section of the enclosing block

Exception Propagation

P
L
/
S
Q
L



```
    EXCEPTION
      WHEN NO_DATA_FOUND THEN
          DBMS_OUTPUT.PUT_LINE('No data error in nested block');
    END;
    lv_junk_num := 3;
EXCEPTION
 WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error Code = '||SQLCODE);
    DBMS_OUTPUT.PUT_LINE('Error Message = '||SQLERRM);
END;
```

```
anonymous block completed
```

```
Error Code = -1422
Error Message = ORA-01422: exact fetch returns more than requested number of rows
```

Commenting Code

- Add comments within code to identify code purpose and processing steps
- Use /*   */ to enclose a multiline comment
- Use -- to add a single or partial line comment

Comment Examples

P
L
/
S
Q
L

```
DECLARE
   ex_prod_update EXCEPTION;  --For UPDATE of no rows
   exception
BEGIN
 /* This block is used to update product descriptions
    Constructed to support the Prod_desc.frm app screen
      Exception raised if no rows updated  */
   UPDATE bb_product
    SET description = 'Mill grinder with 5 grind settings!'
    WHERE idProduct = 30;
   --Check if any rows updated
 IF SQL%NOTFOUND THEN
    RAISE ex_prod_update;
   END IF;
EXCEPTION
   WHEN ex_prod_update THEN
     DBMS_OUTPUT.PUT_LINE('Invalid product id entered');
END;
```

Summary

- Implicit cursors are automatically created for SQL statements
- Explicit cursors are declared
- Cursors allow the processing of a group of rows
- CURSOR FOR Loops simplify cursor coding
- Parameters make cursors more dynamic
- A REF CURSOR acts like a pointer
- BULK processing options can improve performance for queries and DML activity

Summary (continued)

- Add error handlers in the EXCEPTION area to manage Oracle and user-defined errors
- Exception propagation is the flow of error handling processing
- Use comments in code for documentation