P
L
/
S
Q
L

Oracle:
PL/SQL Programming

Chapter 1

Introduction to PL/SQL

Chapter Objectives

P
L
/
S
Q
L

- After completing this lesson, you should be able to understand:
  - PL/SQL and application programming
  - Application models
  - How to locate Oracle resources
  - SQL and PL/SQL tools
  - The databases used in this book
  - SQL SELECT statement and data manipulation syntax

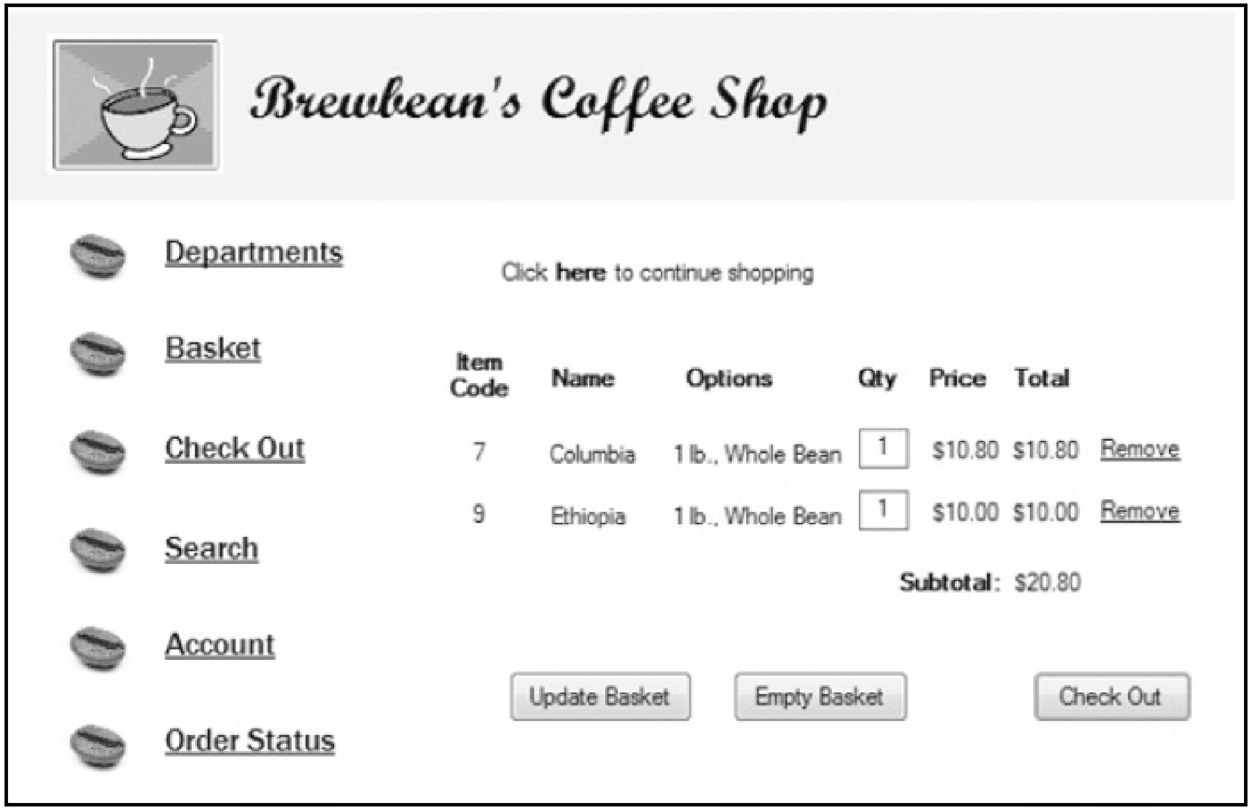
Procedural Languages

- Programming languages allow actions of the end user to be converted to computer instructions
- Procedural languages allow the inclusion of logic processes
- PL/SQL is a procedural language, SQL is not a procedural language

P
L
/
S
Q
L

Application Programming

- Example application screen



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	

Brewbean's Application

- Processing needed to support the shopping cart check out button
  - Verify quantities are > 0
  - Calculate shipping cost
  - Calculate taxes
  - Check/update product inventory
  - Check shopper profile for credit card information

The PL/SQL Language

- Proprietary Oracle language
- Tightly integrated with SQL
- Can increase performance by grouping statements into blocks of code
- Portable to any Oracle platform
- Used within many Oracle tools
- Stored program units can increase security

Application Models

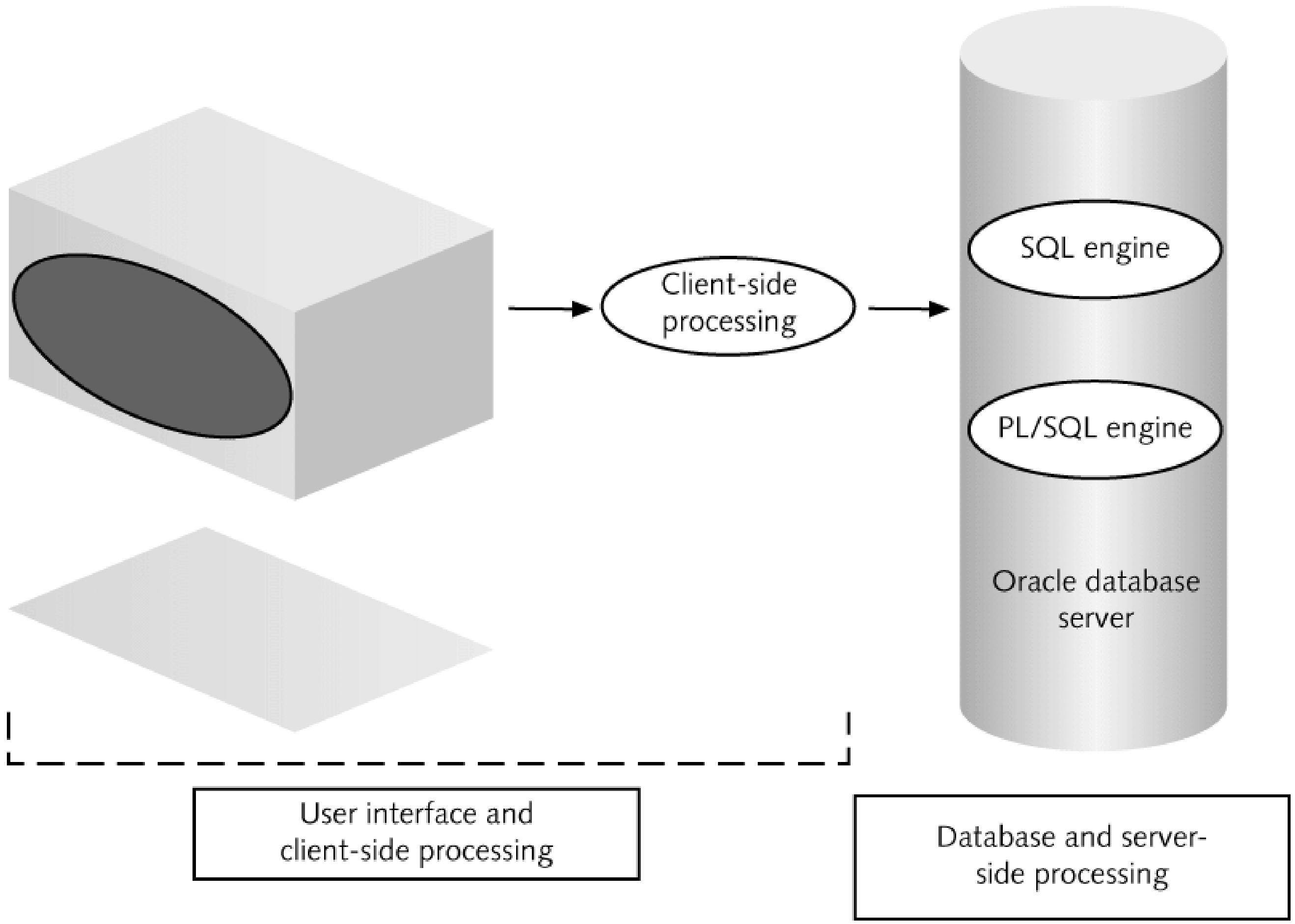
- Three main components
  - User interface or screens
  - Program logic (brains behind the screens)
  - Database
- Most models are based on a two- or three-tier structure

Two-tier Model

P
L
/
S
Q
L

- Commonly referred to as client/server
- Parts of the processing occur both on the user’s computer and the database server
- Named or stored program units are blocks of PL/SQL code saved in the Oracle database to provide server-side processing

Two-tier Diagram

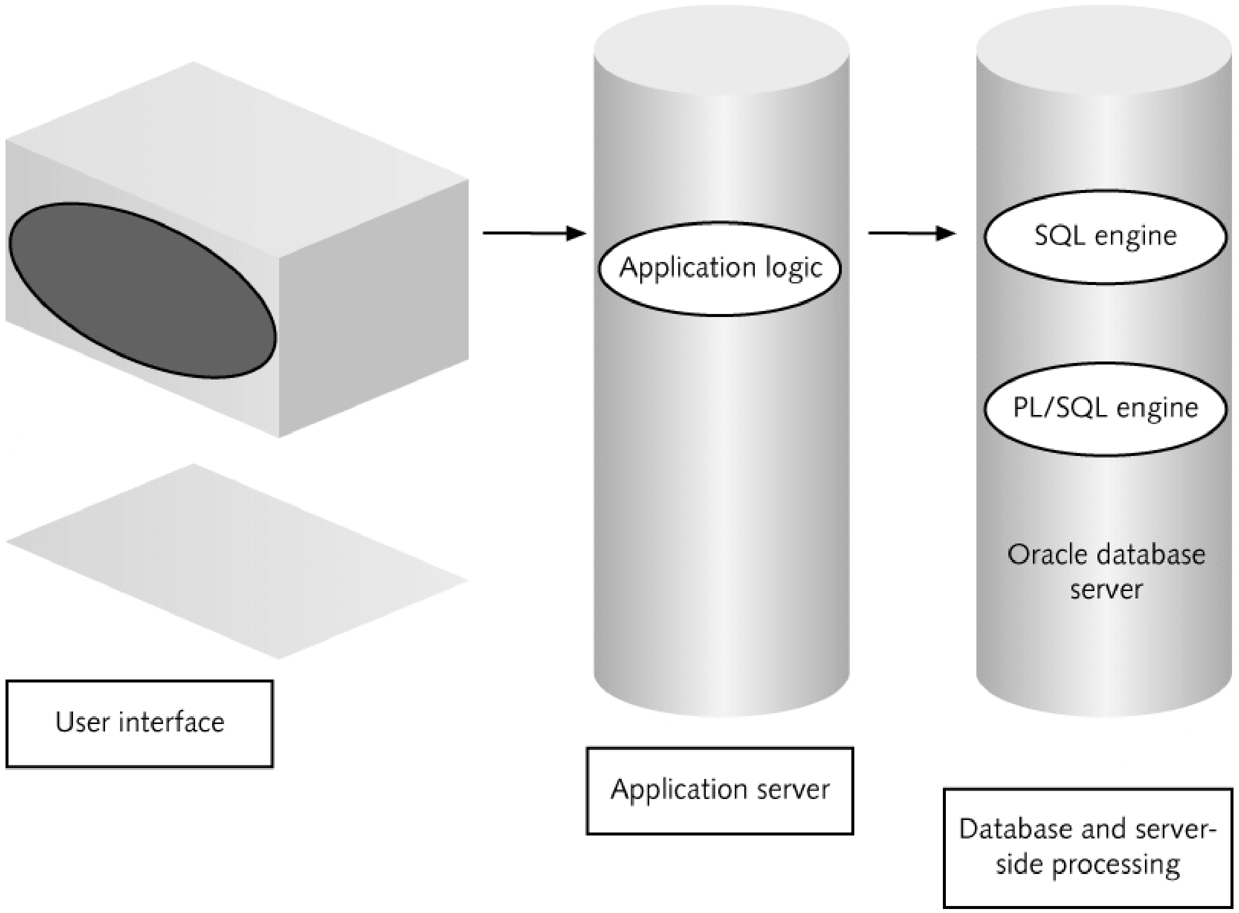

Client-side processing
SQL engine
PL/SQL engine
Oracle database server
User interface and client-side processing
Database and server-side processing

Three-tier Model

- Thin client with no code loaded on the user machine (browser access)
- Middle tier is the application server – Forms server for Oracle
- Last tier is the database server
- Processing load is on the middle and last tier
- Maintenance is simplified

P
L
/
S
Q
L

Three-tier Diagram



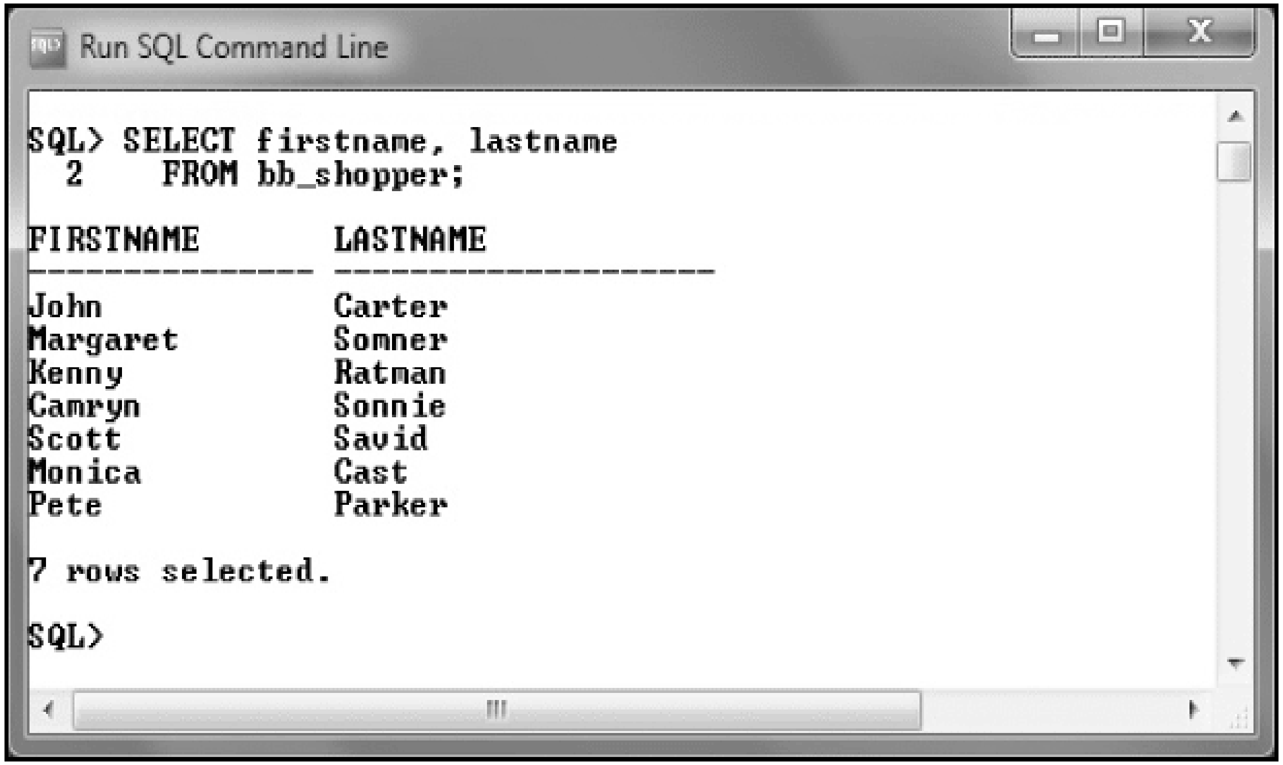
Oracle Documentation

- Oracle Technology Network (OTN): otn.oracle.com
  - Documentation
  - Sample Code
  - Discussion Forums
- User Web sites: PL/SQL Obsession

SQL & PL/SQL Tools

- SQL*Plus
- SQL Developer
  - Appendix B
- Other software introduced in appendices
  - TOAD
  - SQL Navigator

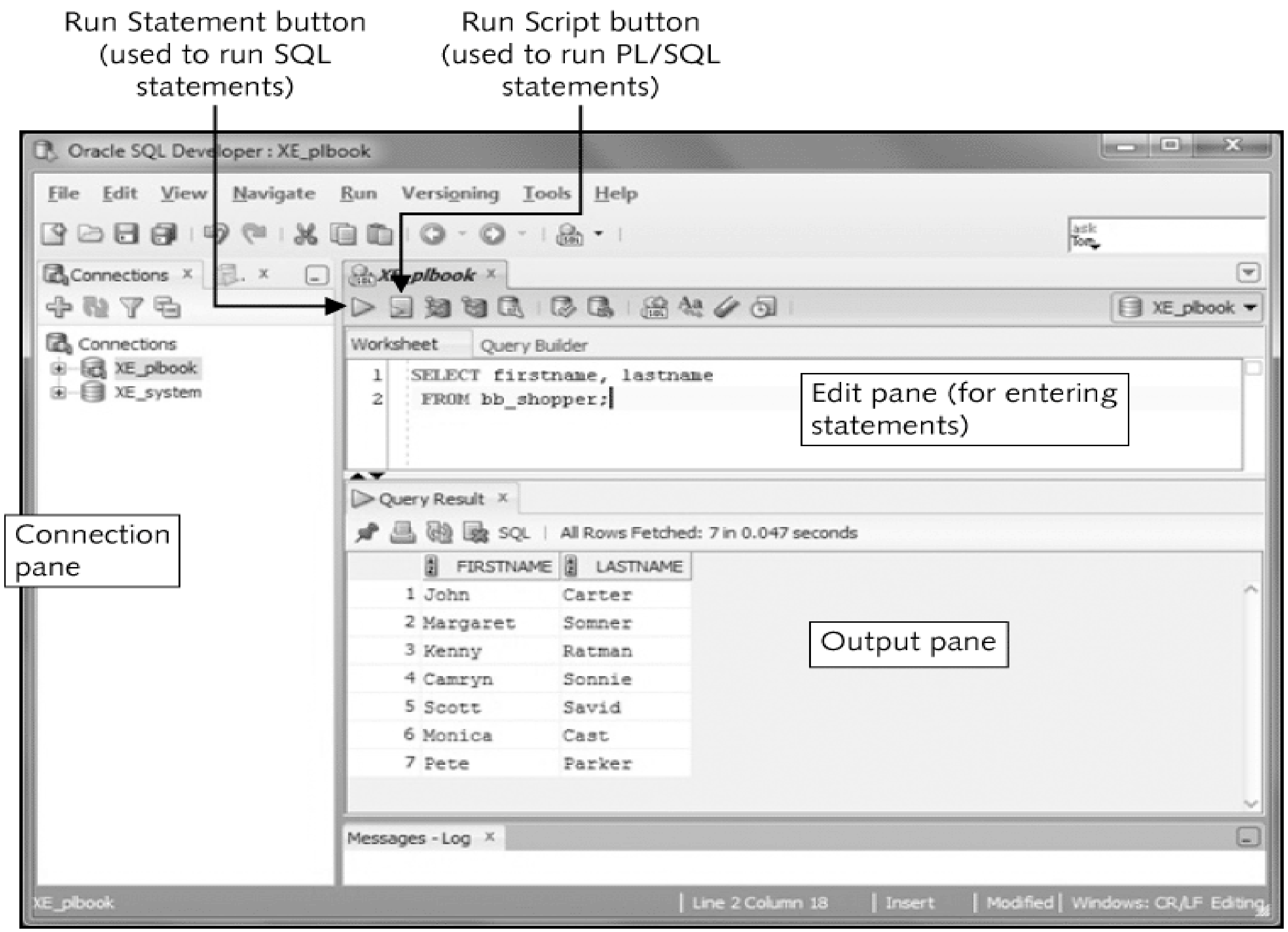
SQL*Plus Client Interface



```
SQL> SELECT firstname, lastname
  2    FROM bb_shopper;

FIRSTNAME       LASTNAME
--------------- --------------------
John            Carter
Margaret        Somner
Kenny           Ratman
Camryn          Sonnie
Scott           Savid
Monica          Cast
Pete            Parker

7 rows selected.

SQL>
```

SQL Developer

P
L
/
S
Q
L


Run Statement button (used to run SQL statements)
Run Script button (used to run PL/SQL statements)
Edit pane (for entering statements)
Connection pane
Output pane
Oracle SQL Developer : XE_plbook
File Edit View Navigate Run Versioning Tools Help
Connections
XE_plbook
XE_system
Worksheet
Query Builder
Query Result
SQL
Messages - Log
Line 2 Column 18
Insert
Modified
Windows: CR/LF Editing

```
SELECT firstname, lastname
 FROM bb_shopper;
```

All Rows Fetched: 7 in 0.047 seconds

```
   FIRSTNAME   LASTNAME
 1 John        Carter
 2 Margaret    Somner
 3 Kenny       Ratman
 4 Camryn      Sonnie
 5 Scott       Savid
 6 Monica      Cast
 7 Pete        Parker
```

Databases Used

P
L
/
S
Q
L

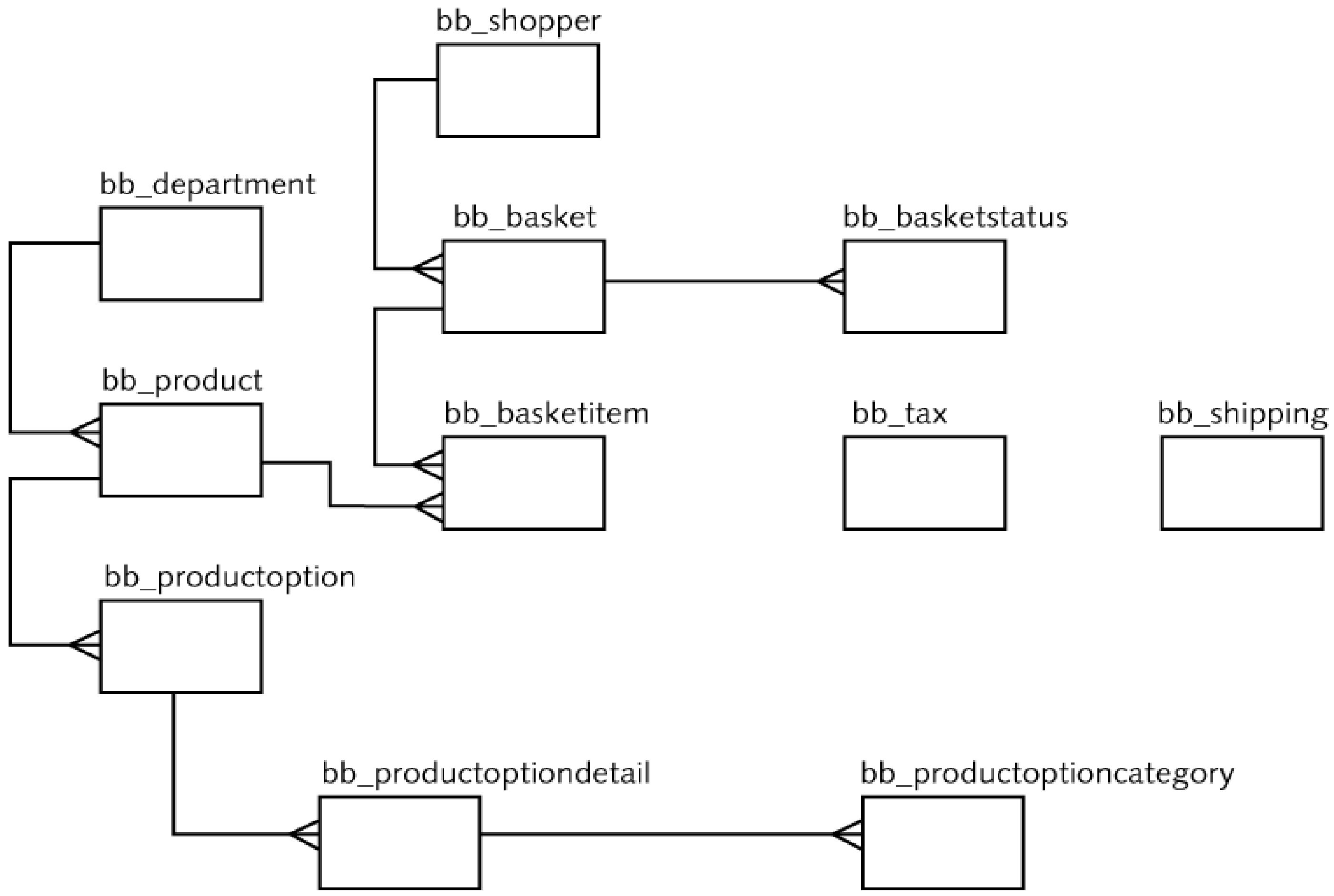
- Brewbean’s Company
  - In text examples
  - Assignments
- DoGood Donor
  - Assignments
- More Movie Rentals
  - Case Projects

The Brewbean's Company

P
L
/
S
Q
L

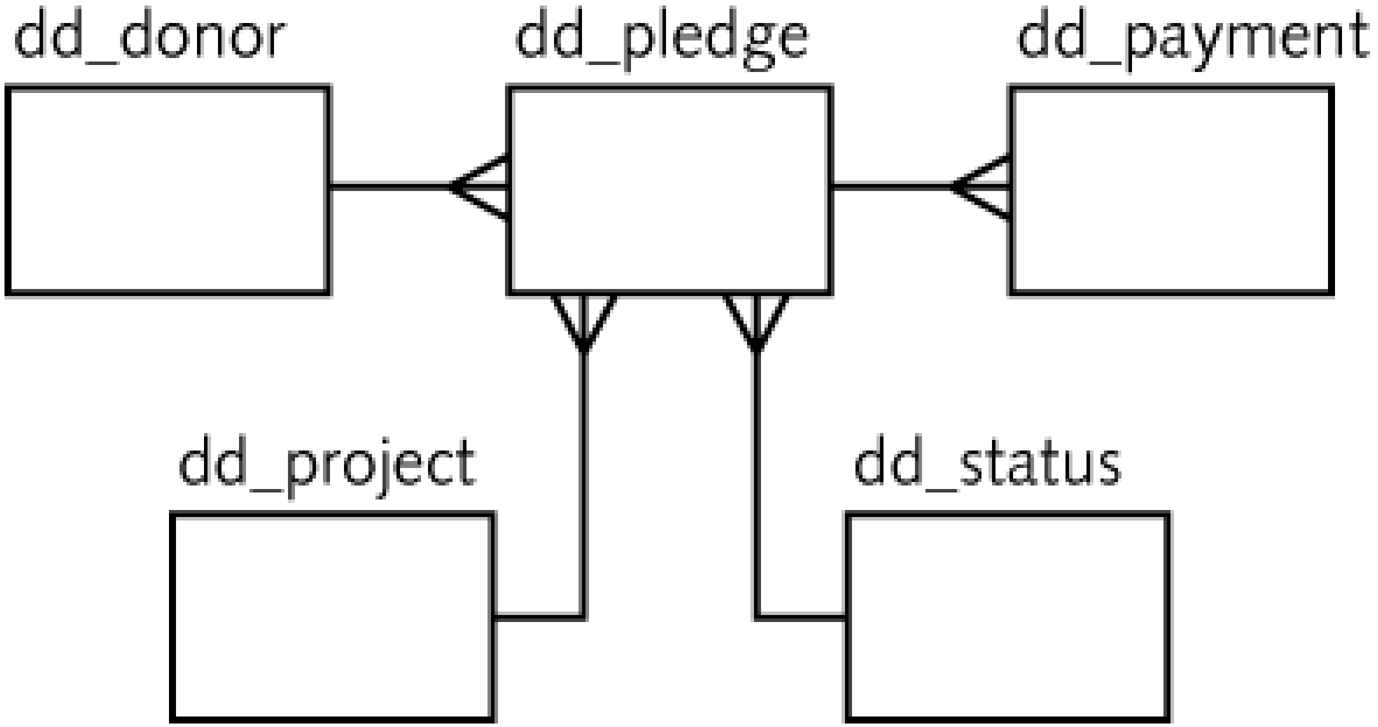
- Retails coffee and brewing equipment via the Internet, phone, and stores
- Used in chapter explanations, examples, and exercises
- Databases create script provided for each chapter

ERD for Brewbean's DB



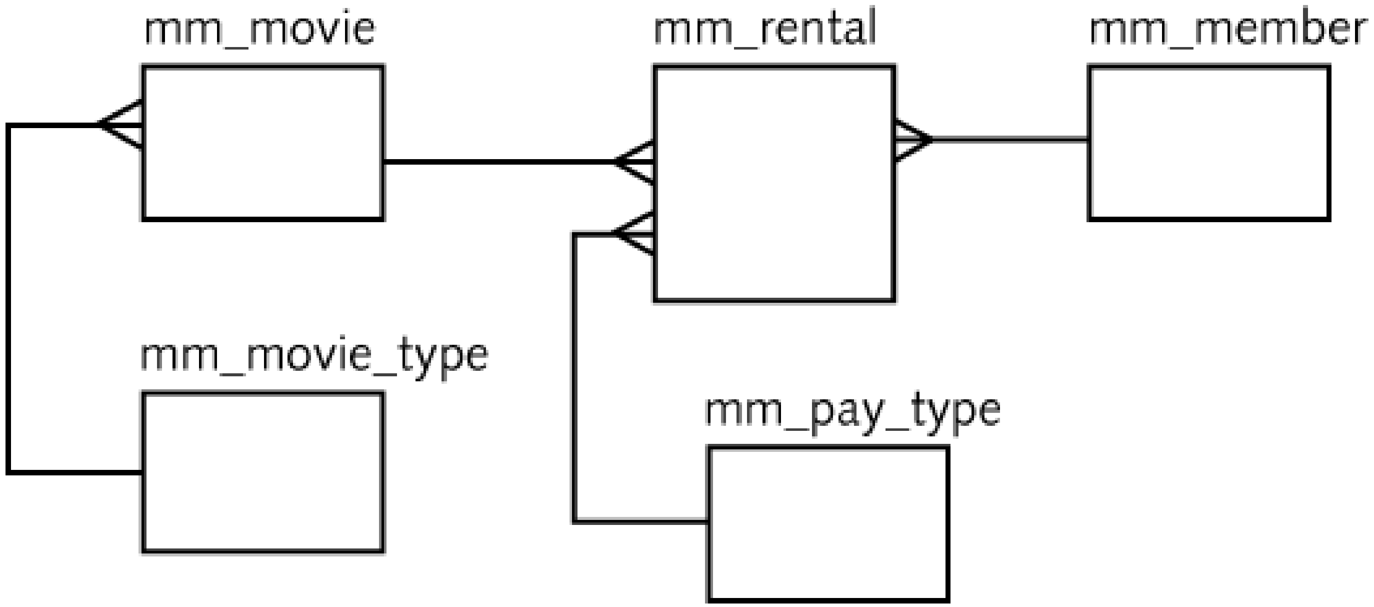
DoGood Donor ERD

- Track donation, pledges, and payments



More Movies ERD

- Movie rental company used in an ongoing case study

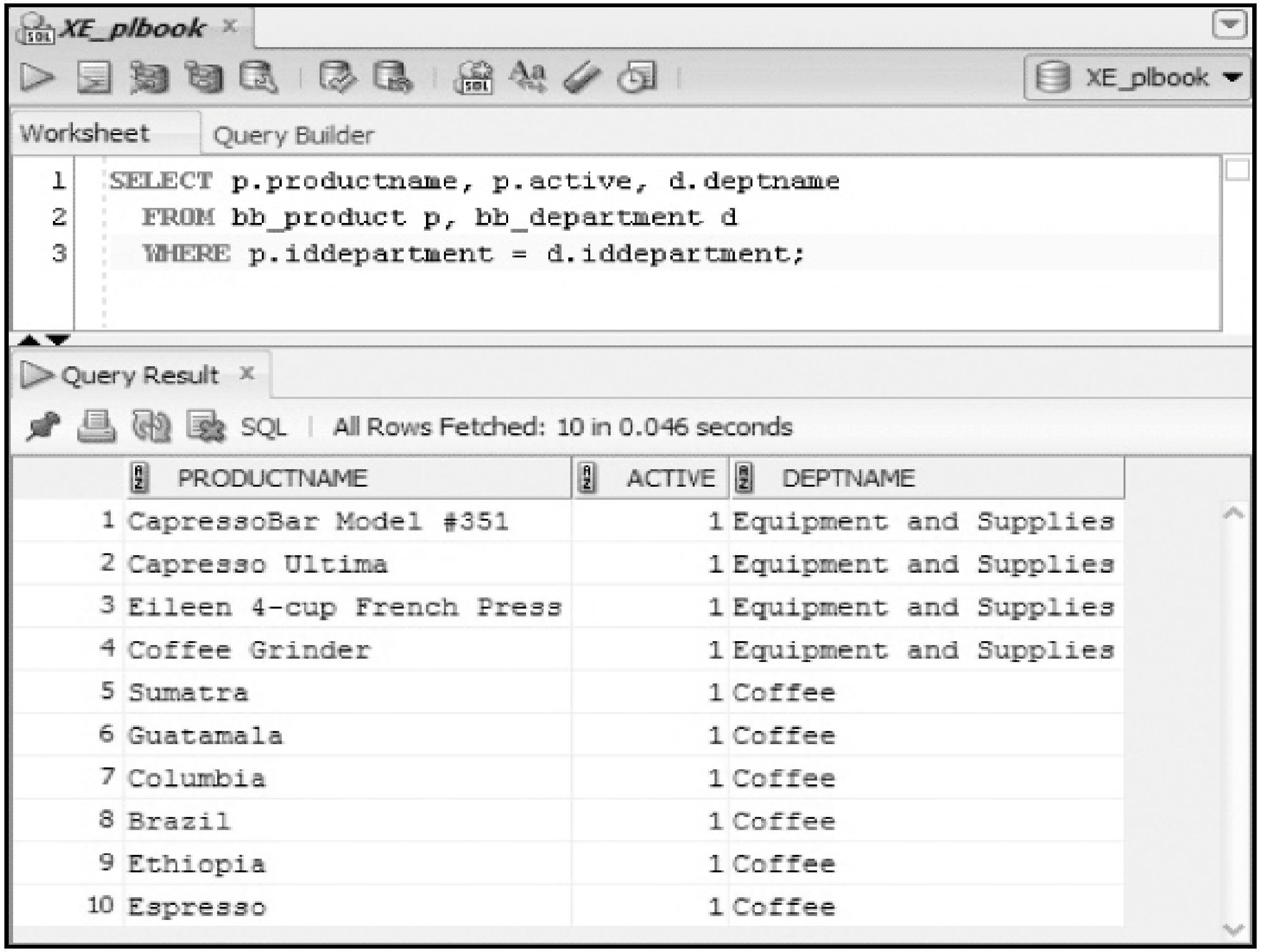


SQL Query Syntax

```
SELECT <columns>
   FROM <tables, views>
   WHERE <conditions>
   GROUP BY <columns>
   HAVING <aggregation conditions>
   ORDER BY <columns>;
```

Traditional Join

P
L
/
S
Q
L



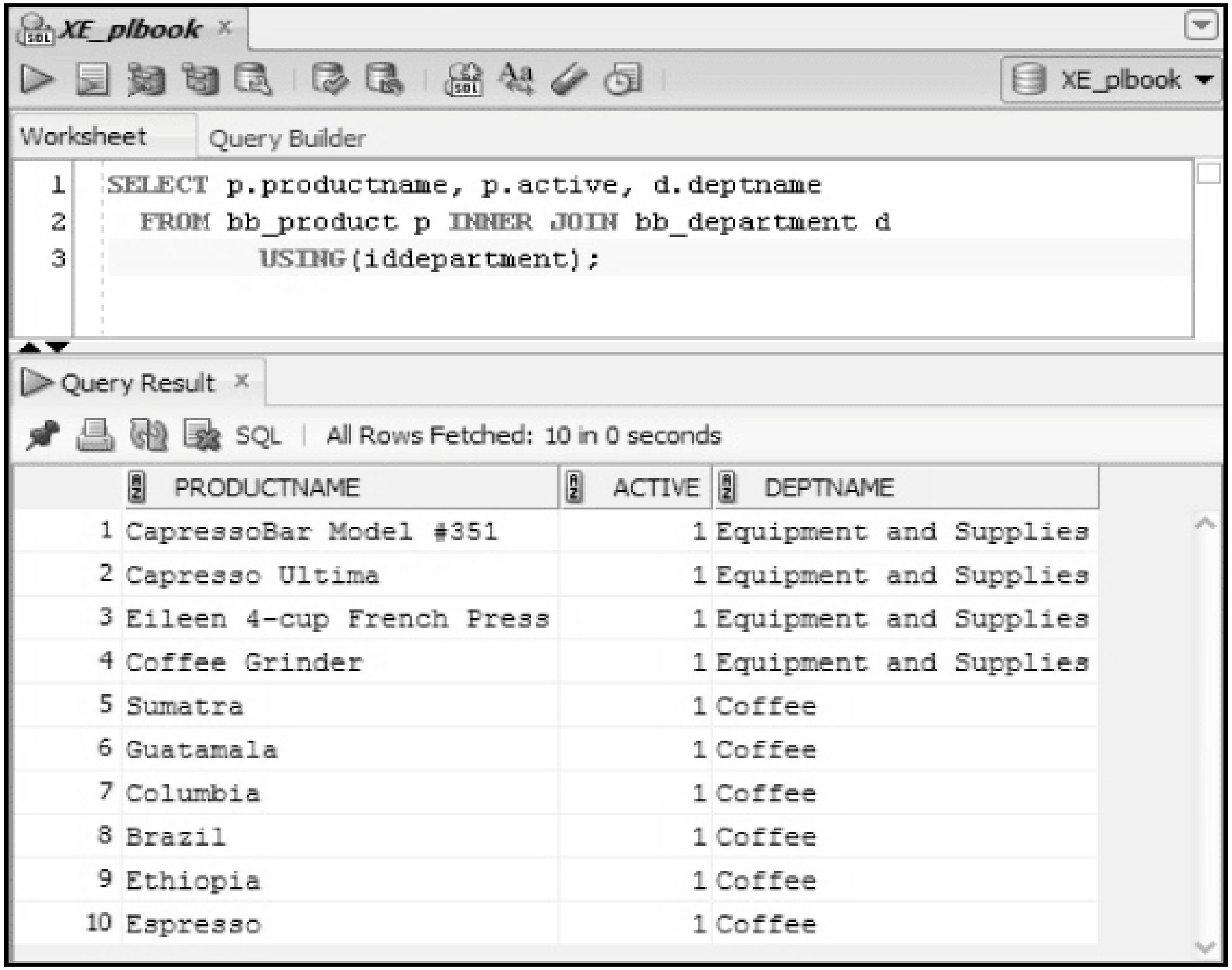
```
SELECT p.productname, p.active, d.deptname
  FROM bb_product p, bb_department d
  WHERE p.iddepartment = d.iddepartment;
```

All Rows Fetched: 10 in 0.046 seconds

	PRODUCTNAME	ACTIVE	DEPTNAME
1	CapressoBar Model #351	1	Equipment and Supplies
2	Capresso Ultima	1	Equipment and Supplies
3	Eileen 4-cup French Press	1	Equipment and Supplies
4	Coffee Grinder	1	Equipment and Supplies
5	Sumatra	1	Coffee
6	Guatamala	1	Coffee
7	Columbia	1	Coffee
8	Brazil	1	Coffee
9	Ethiopia	1	Coffee
10	Espresso	1	Coffee

P
L
/
S
Q
L

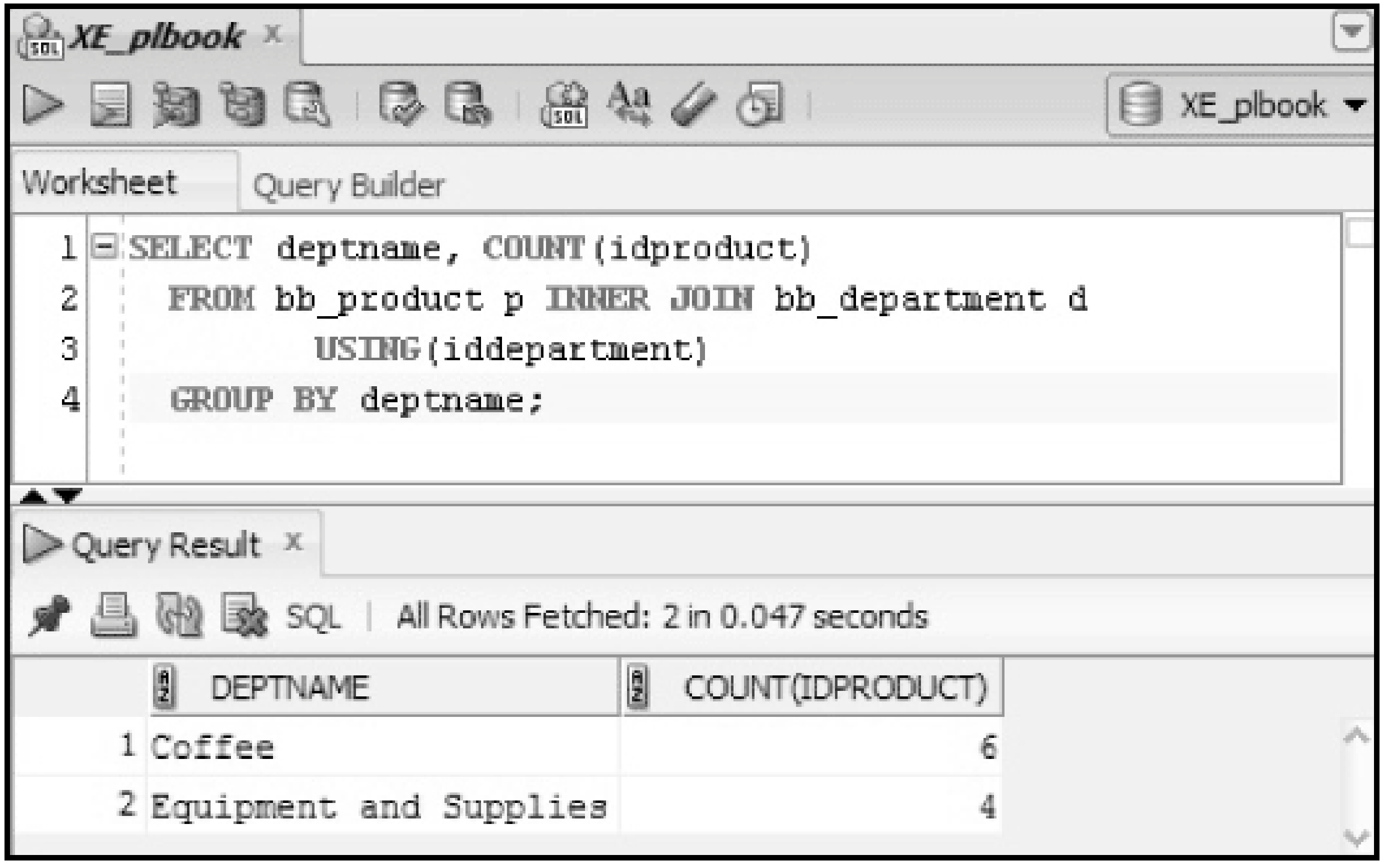
ANSI Join



```
SELECT p.productname, p.active, d.deptname
  FROM bb_product p INNER JOIN bb_department d
         USING(iddepartment);
```

	PRODUCTNAME	ACTIVE	DEPTNAME
1	CapressoBar Model #351	1	Equipment and Supplies
2	Capresso Ultima	1	Equipment and Supplies
3	Eileen 4-cup French Press	1	Equipment and Supplies
4	Coffee Grinder	1	Equipment and Supplies
5	Sumatra	1	Coffee
6	Guatamala	1	Coffee
7	Columbia	1	Coffee
8	Brazil	1	Coffee
9	Ethiopia	1	Coffee
10	Espresso	1	Coffee

Aggregate function

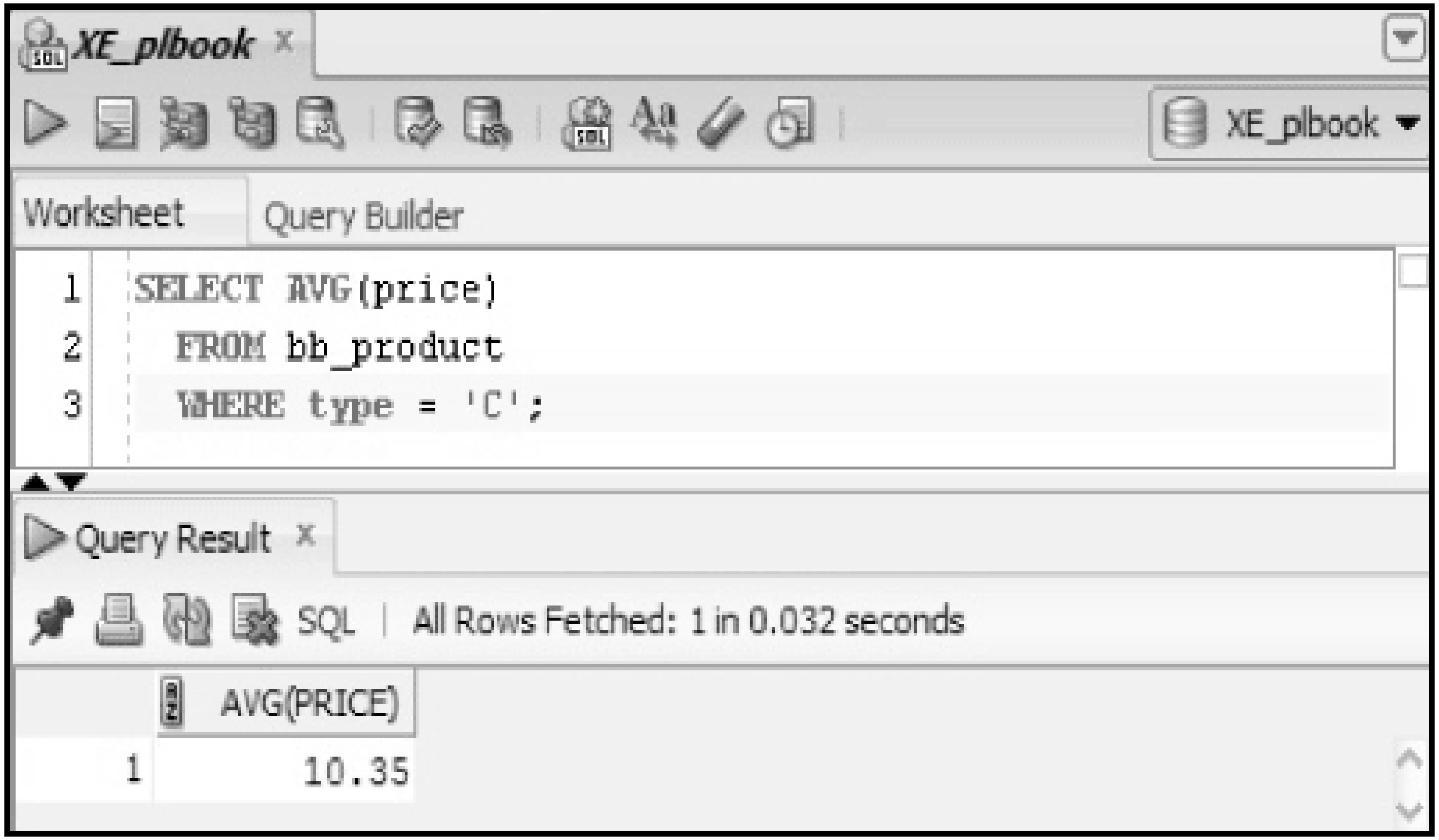


```
SELECT deptname, COUNT(idproduct)
  FROM bb_product p INNER JOIN bb_department d
         USING(iddepartment)
  GROUP BY deptname;
```

All Rows Fetched: 2 in 0.047 seconds

```
  DEPTNAME                  COUNT(IDPRODUCT)
1 Coffee                                   6
2 Equipment and Supplies                   4
```

WHERE clause filter

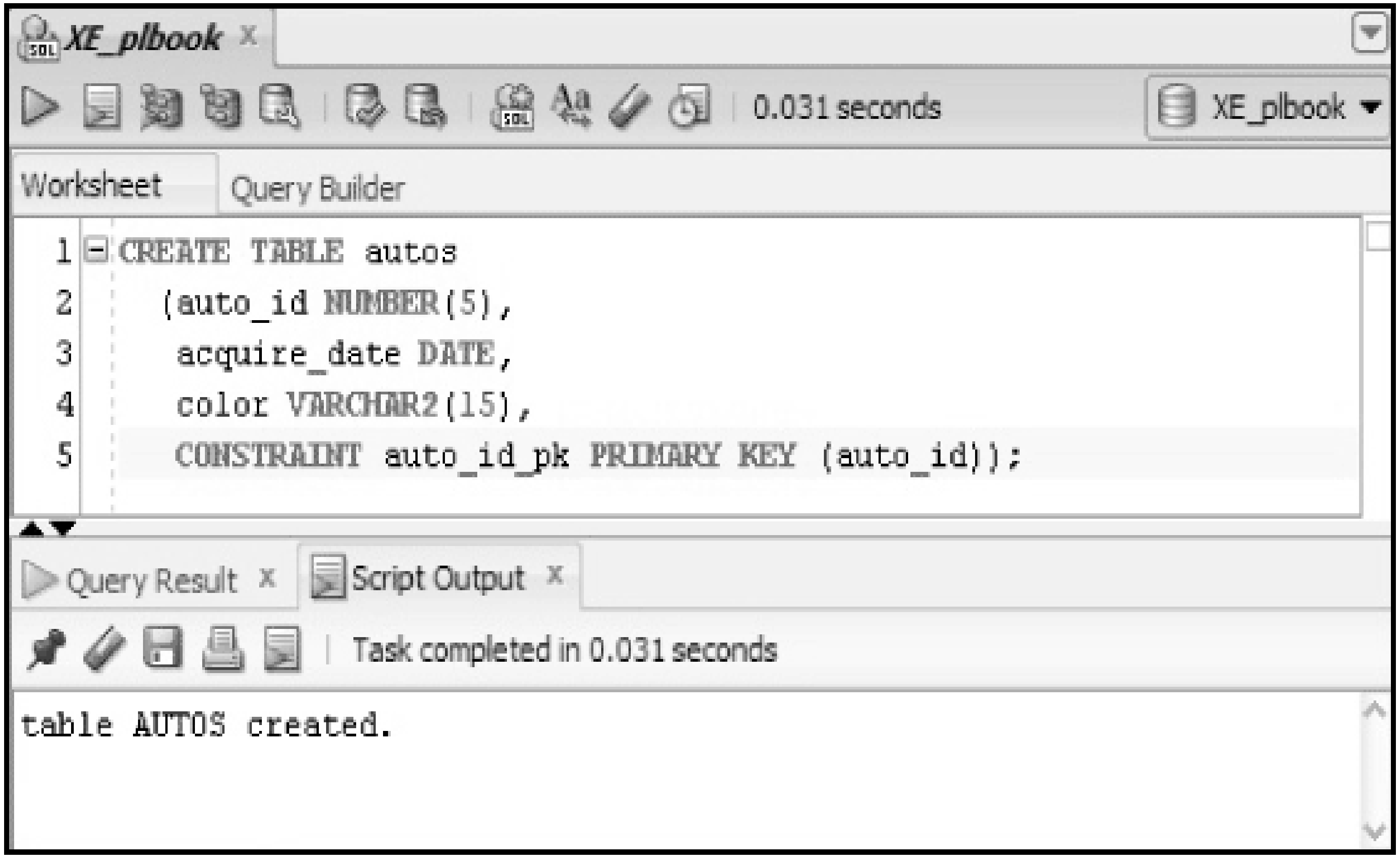


```
SELECT AVG(price)
  FROM bb_product
  WHERE type = 'C';
```

All Rows Fetched: 1 in 0.032 seconds

	AVG(PRICE)
1	10.35

Creating Tables

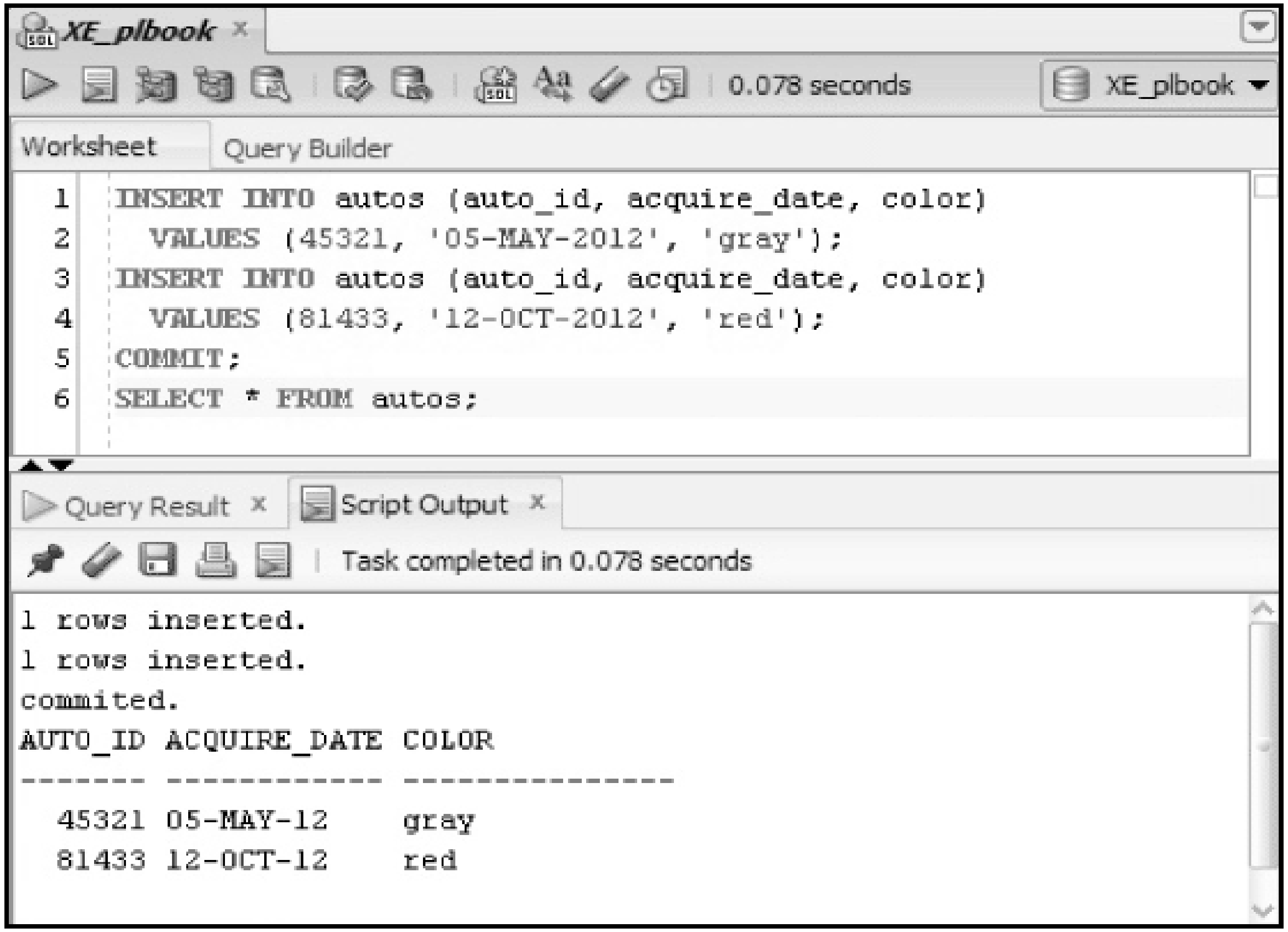


```
CREATE TABLE autos
  (auto_id NUMBER(5),
   acquire_date DATE,
   color VARCHAR2(15),
   CONSTRAINT auto_id_pk PRIMARY KEY (auto_id));
```

```
table AUTOS created.
```

DML - Insert

P
L
/
S
Q
L

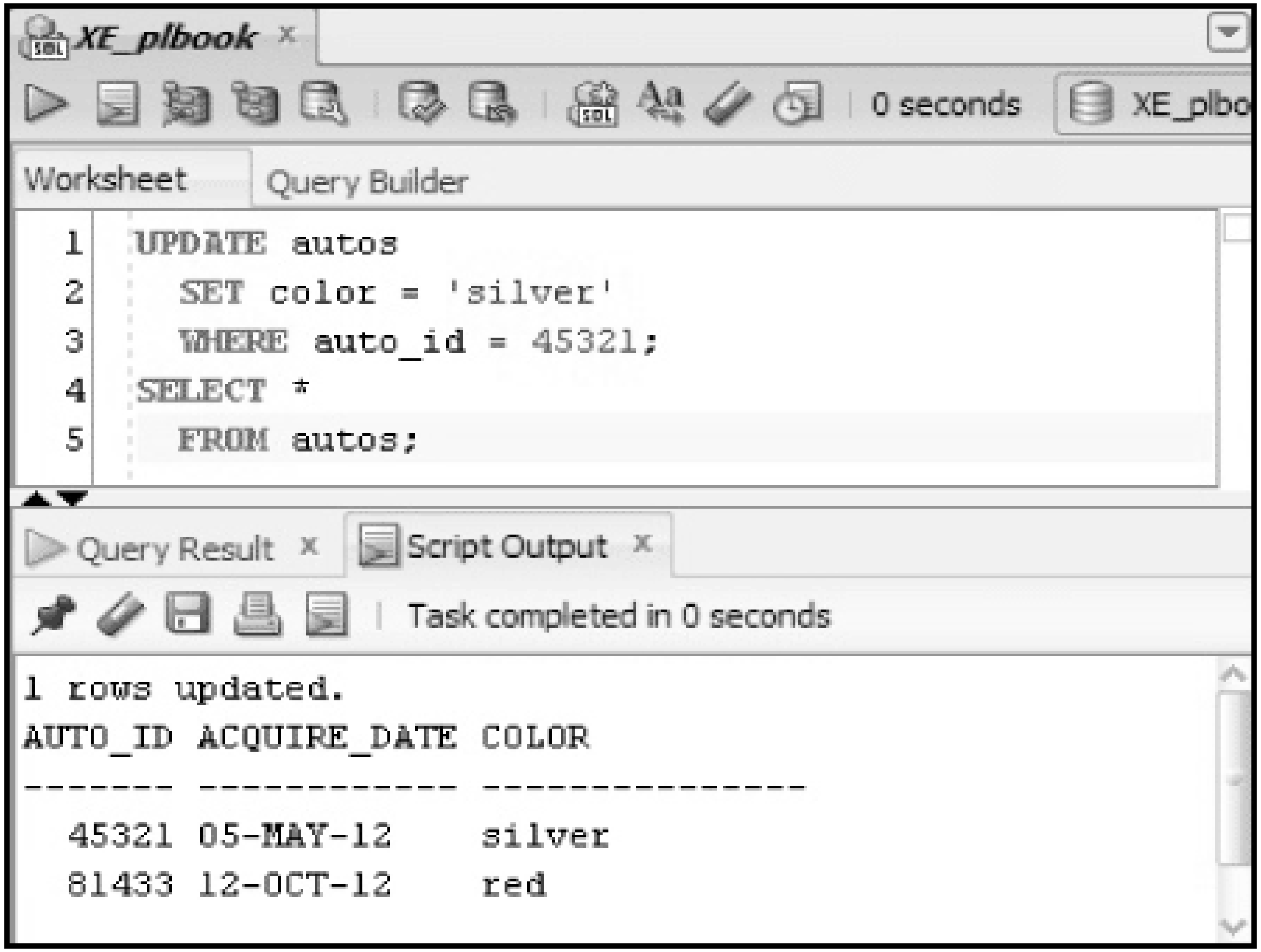


```
INSERT INTO autos (auto_id, acquire_date, color)
  VALUES (45321, '05-MAY-2012', 'gray');
INSERT INTO autos (auto_id, acquire_date, color)
  VALUES (81433, '12-OCT-2012', 'red');
COMMIT;
SELECT * FROM autos;
```

```
1 rows inserted.
1 rows inserted.
commited.
AUTO_ID ACQUIRE_DATE COLOR
------- ------------ ---------------
  45321 05-MAY-12    gray
  81433 12-OCT-12    red
```

DML - Update

P
L
/
S
Q
L

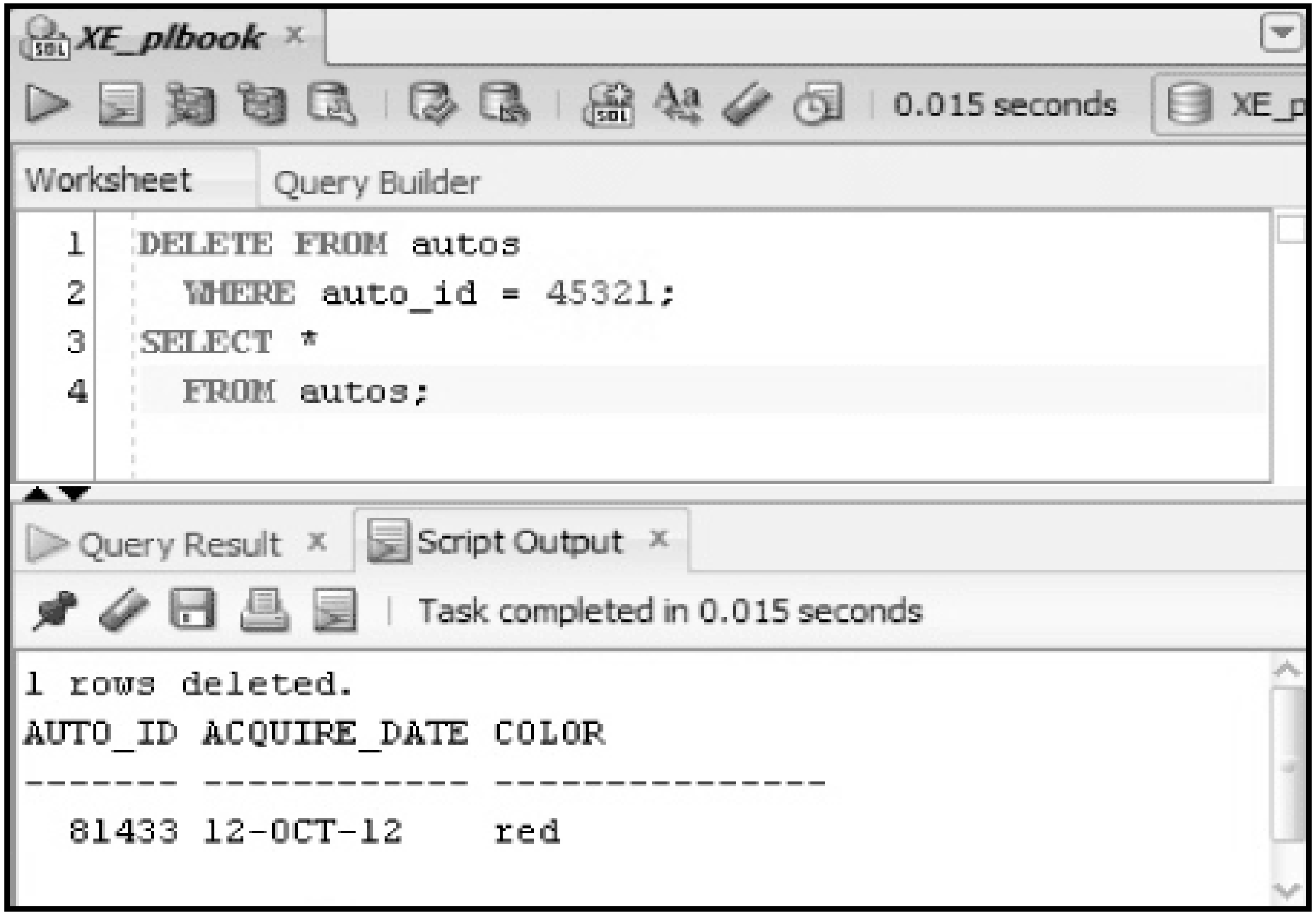


```
UPDATE autos
  SET color = 'silver'
  WHERE auto_id = 45321;
SELECT *
  FROM autos;
```

```
1 rows updated.
AUTO_ID ACQUIRE_DATE COLOR
------- ------------ ---------------
  45321 05-MAY-12    silver
  81433 12-OCT-12    red
```

P
L
/
S
Q
L

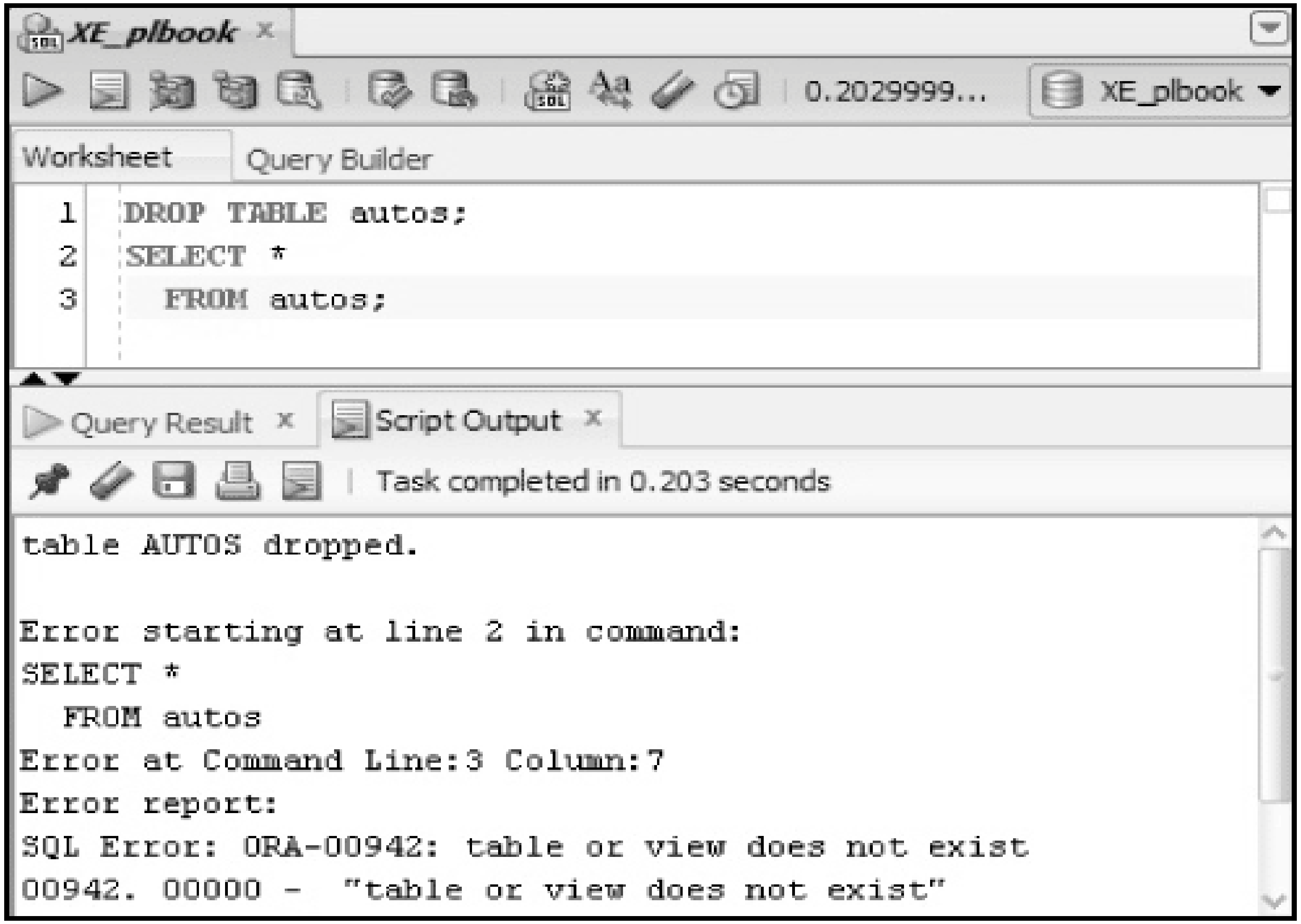
DML - Delete



```
DELETE FROM autos
  WHERE auto_id = 45321;
SELECT *
  FROM autos;
```

```
1 rows deleted.
AUTO_ID ACQUIRE_DATE COLOR
------- ------------ ---------------
  81433 12-OCT-12    red
```

Drop Table



```
DROP TABLE autos;
SELECT *
  FROM autos;
```

```
table AUTOS dropped.

Error starting at line 2 in command:
SELECT *
  FROM autos
Error at Command Line:3 Column:7
Error report:
SQL Error: ORA-00942: table or view does not exist
00942. 00000 -  "table or view does not exist"
```

Review to prepare

- Review SQL statement syntax
- Explore the Brewbean’s database

P
L
/
S
Q
L

Oracle:
PL/SQL Programming

Chapter 2

Basic PL/SQL Block Structures

Chapter Objectives

- After completing this lesson, you should be able to understand:
  - Programming fundamentals
  - PL/SQL blocks
  - How to define and declare variables
  - How to initialize and manage variable values
  - The NOT NULL and CONSTANT variable options

Chapter Objectives (continued)

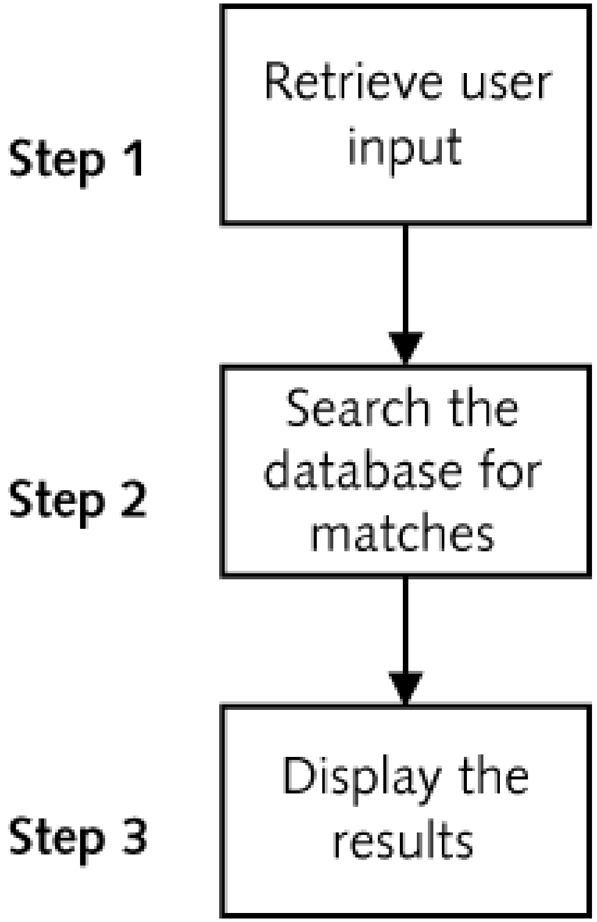
- After completing this lesson, you should be able to understand (continued):
  - How to perform calculations with variables
  - The use of SQL single-row functions in PL/SQL statements
  - Decision structures: IF-THEN and CASE
  - Looping actions: basic, FOR and WHILE
  - CONTINUE statements
  - Nested Statements

Program Logic Flow

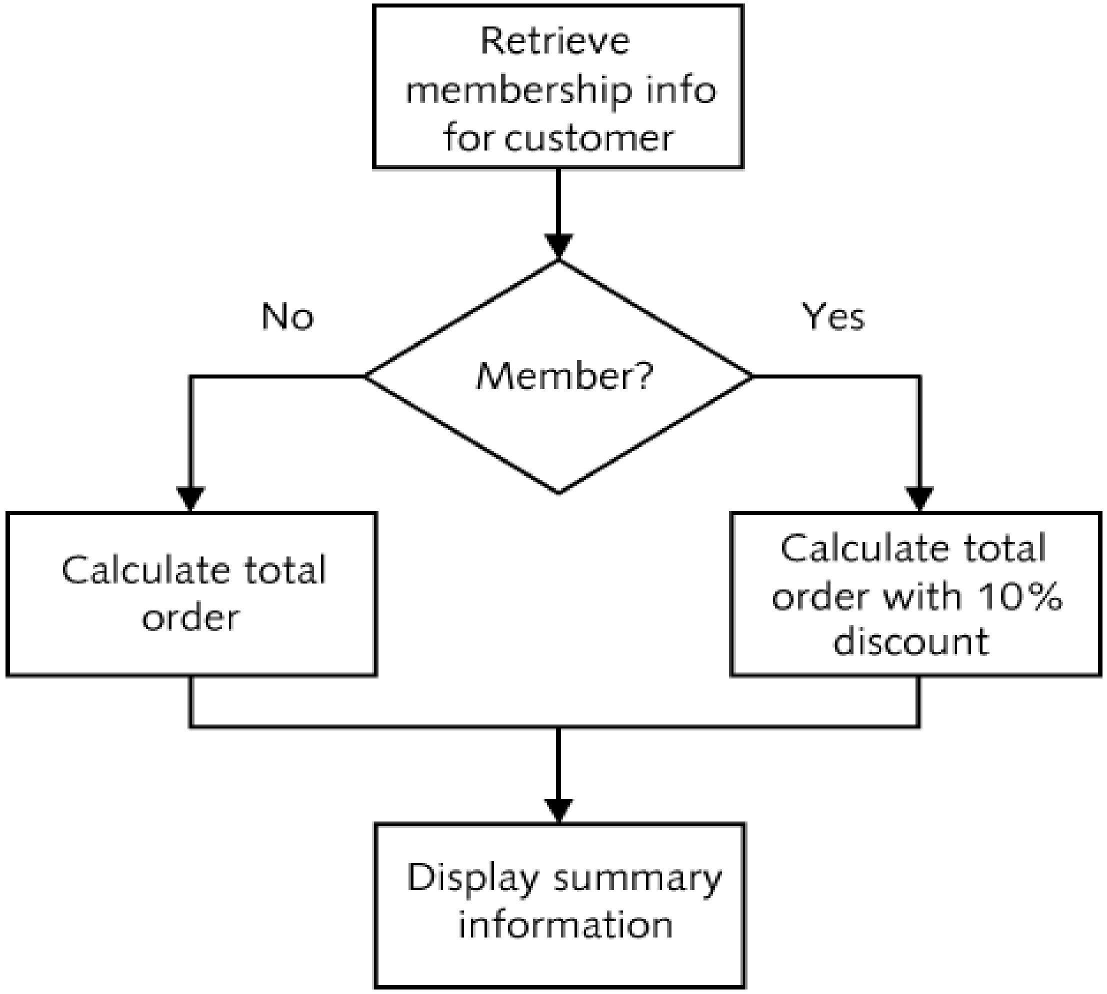
P
L
/
S
Q
L

- Identify sequence of actions needed prior to coding
- Use a flowchart to visually represent the sequence of actions

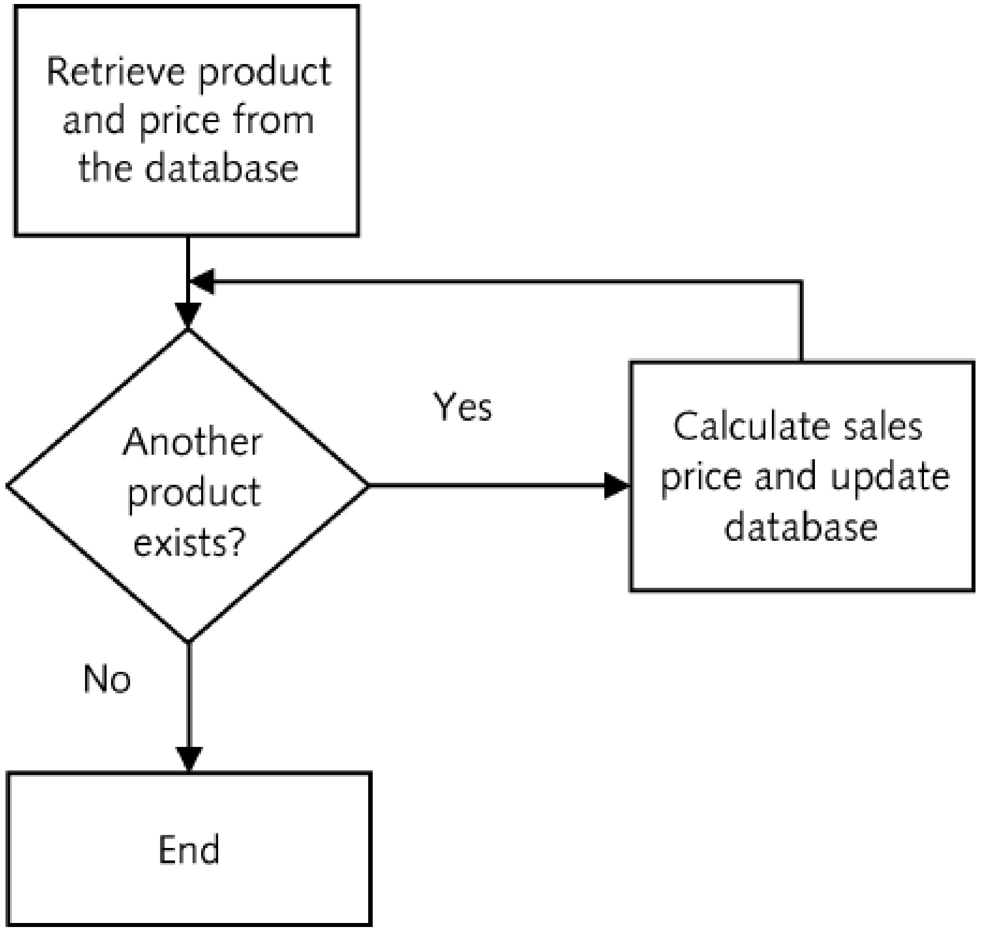
Flowcharting - Search for Coffee Products



Decision Structures


Retrieve membership info for customer
No
Member?
Yes
Calculate total order
Calculate total order with 10% discount
Display summary information

Looping Structures

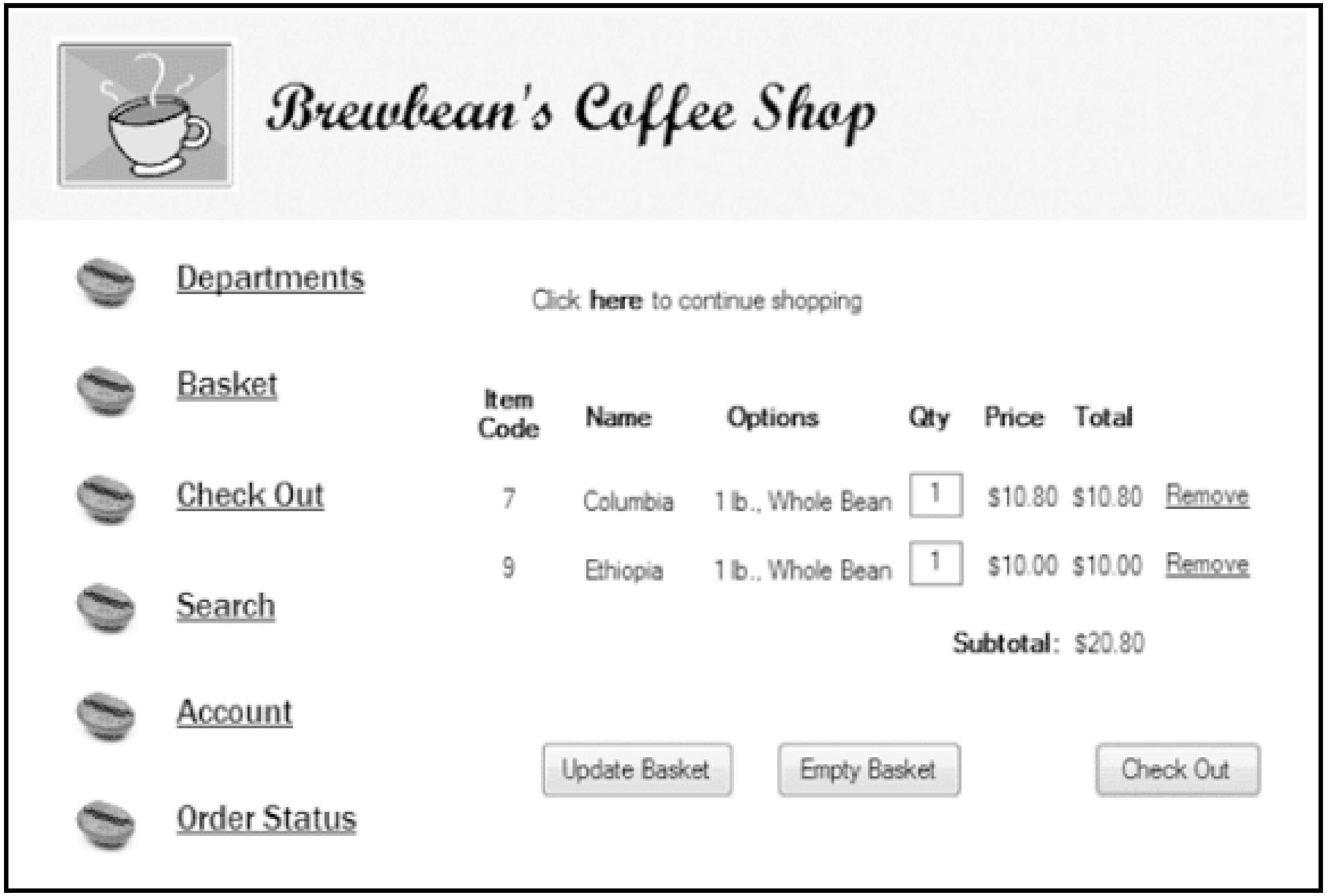

Retrieve product and price from the database
Another product exists?
Yes
Calculate sales price and update database
No
End

PL/SQL Block Questions

P
L
/
S
Q
L

- What is a block?
- What are the different segments of a block?
- How does data get into a block?
- How are different data types handled?

Brewbean's Challenge



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

PL/SQL Block Structure

- **DECLARE** – create variables, cursors, and types
- **BEGIN** – SQL, logic, loops, assignment statements
- **EXCEPTION** – error handling
- **END** – close the block

Variable Names

- Begin with alpha character
- Up to 30 characters
- Can contain upper and lowercase letters, numbers, _ , $, #

Scalar Variable Data Types

P
L
/
S
Q
L

- **Character** – CHAR(n)
  VARCHAR2(n)
- **Numeric** – NUMBER(p,s)
- **Date** – DATE
- **Boolean** – BOOLEAN

Note: Only holds a single value

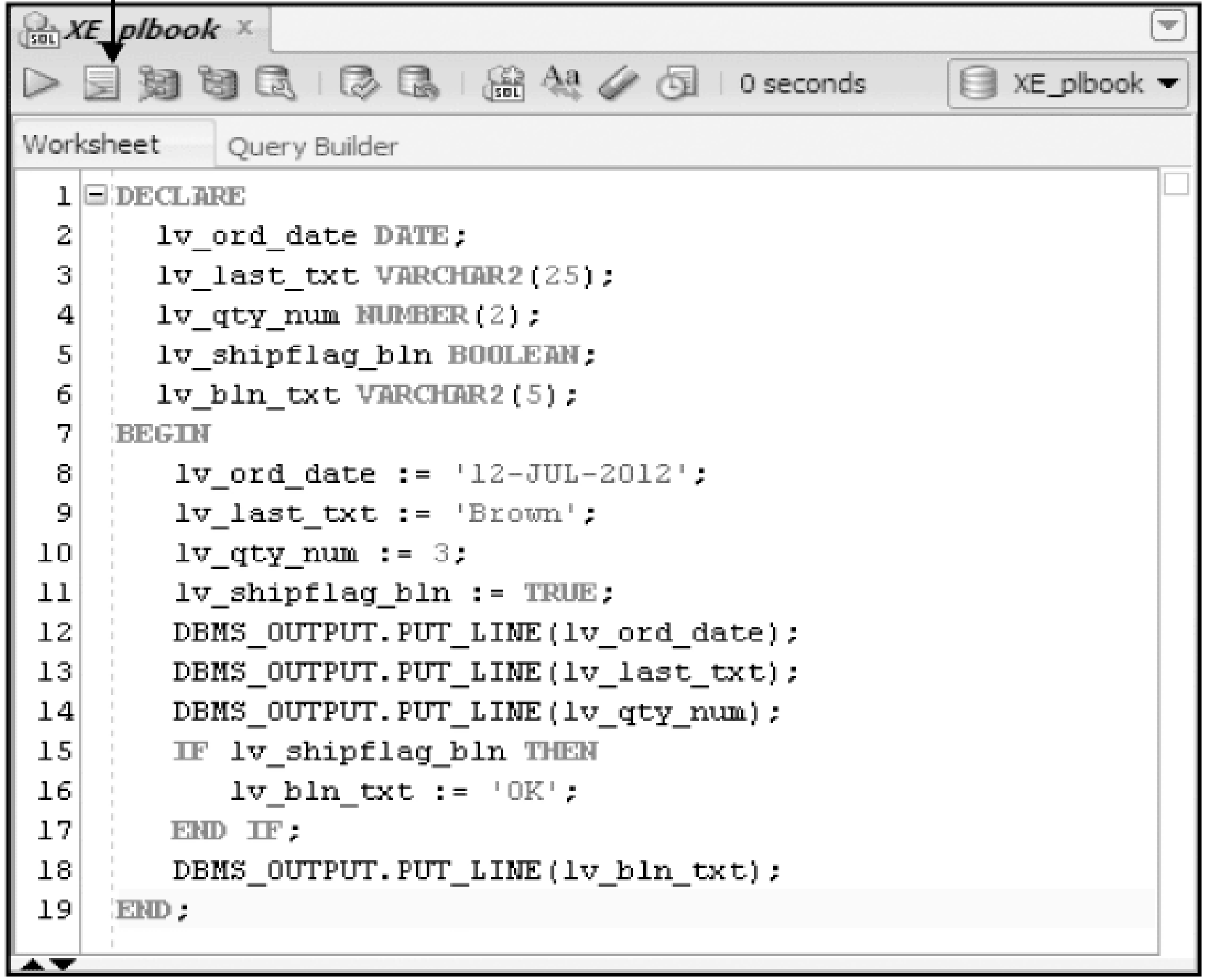
Example Scalar Declarations

```
DECLARE
  lv_ord_date DATE;
  lv_last_txt VARCHAR2(25);
  lv_qty_num NUMBER(2);
  lv_shipflag_bln BOOLEAN;
BEGIN
 ---- PL/SQL executable statements ----
END;
```

Note: Minimum requirements are variable name and data type

Test Variables

Run Script button



```
DECLARE
   lv_ord_date DATE;
   lv_last_txt VARCHAR2(25);
   lv_qty_num NUMBER(2);
   lv_shipflag_bln BOOLEAN;
   lv_bln_txt VARCHAR2(5);
BEGIN
    lv_ord_date := '12-JUL-2012';
    lv_last_txt := 'Brown';
    lv_qty_num := 3;
    lv_shipflag_bln := TRUE;
    DBMS_OUTPUT.PUT_LINE(lv_ord_date);
    DBMS_OUTPUT.PUT_LINE(lv_last_txt);
    DBMS_OUTPUT.PUT_LINE(lv_qty_num);
    IF lv_shipflag_bln THEN
       lv_bln_txt := 'OK';
    END IF;
    DBMS_OUTPUT.PUT_LINE(lv_bln_txt);
END;
```

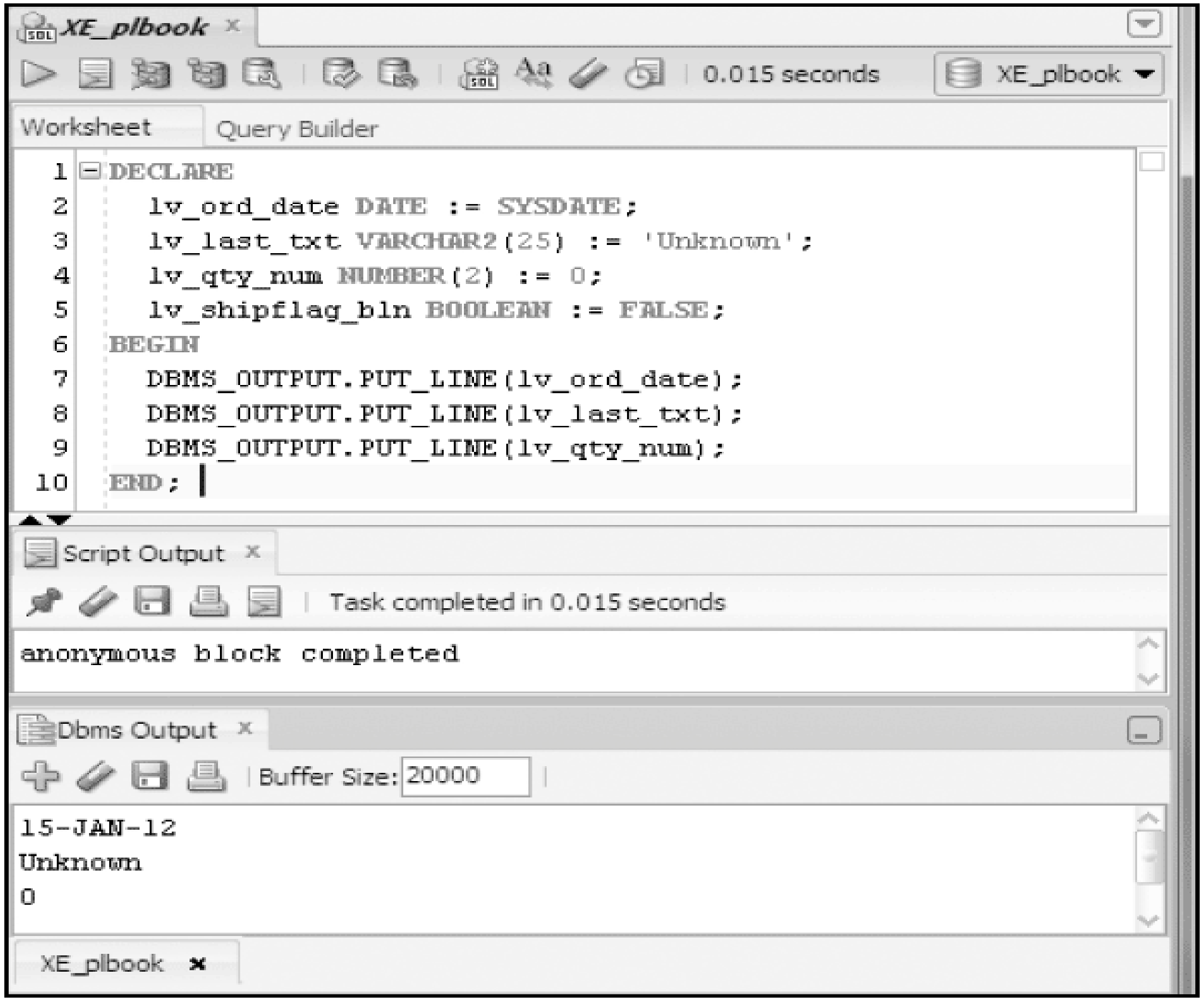
P
L
/
S
Q
L

Variable Initialization

- Set a variable value when the variable is created

```
DECLARE
  lv_ord_date DATE := SYSDATE;
  lv_last_txt VARCHAR2(25) := 'Unknown';
  lv_qty_num NUMBER(2) := 0;
  lv_shipflag_bln BOOLEAN := 'FALSE';
BEGIN
 ---- PL/SQL executable statements ----
END;
```

Test Variable Initialization



```
DECLARE
   lv_ord_date DATE := SYSDATE;
   lv_last_txt VARCHAR2(25) := 'Unknown';
   lv_qty_num NUMBER(2) := 0;
   lv_shipflag_bln BOOLEAN := FALSE;
BEGIN
  DBMS_OUTPUT.PUT_LINE(lv_ord_date);
  DBMS_OUTPUT.PUT_LINE(lv_last_txt);
  DBMS_OUTPUT.PUT_LINE(lv_qty_num);
END;
```

```
anonymous block completed
```

```
15-JAN-12
Unknown
0
```

Variable Declaration Options

- NOT NULL – the variable must always contain a value
- CONSTANT – the variable value can not be changed in the block

```
DECLARE
  lv_shipcntry_txt VARCHAR2(15) NOT NULL := 'US';
  lv_taxrate_num CONSTANT NUMBER(2,2) := .06;
BEGIN
 ---- PL/SQL executable statements ----
END;
```

Calculations with Scalar Variables

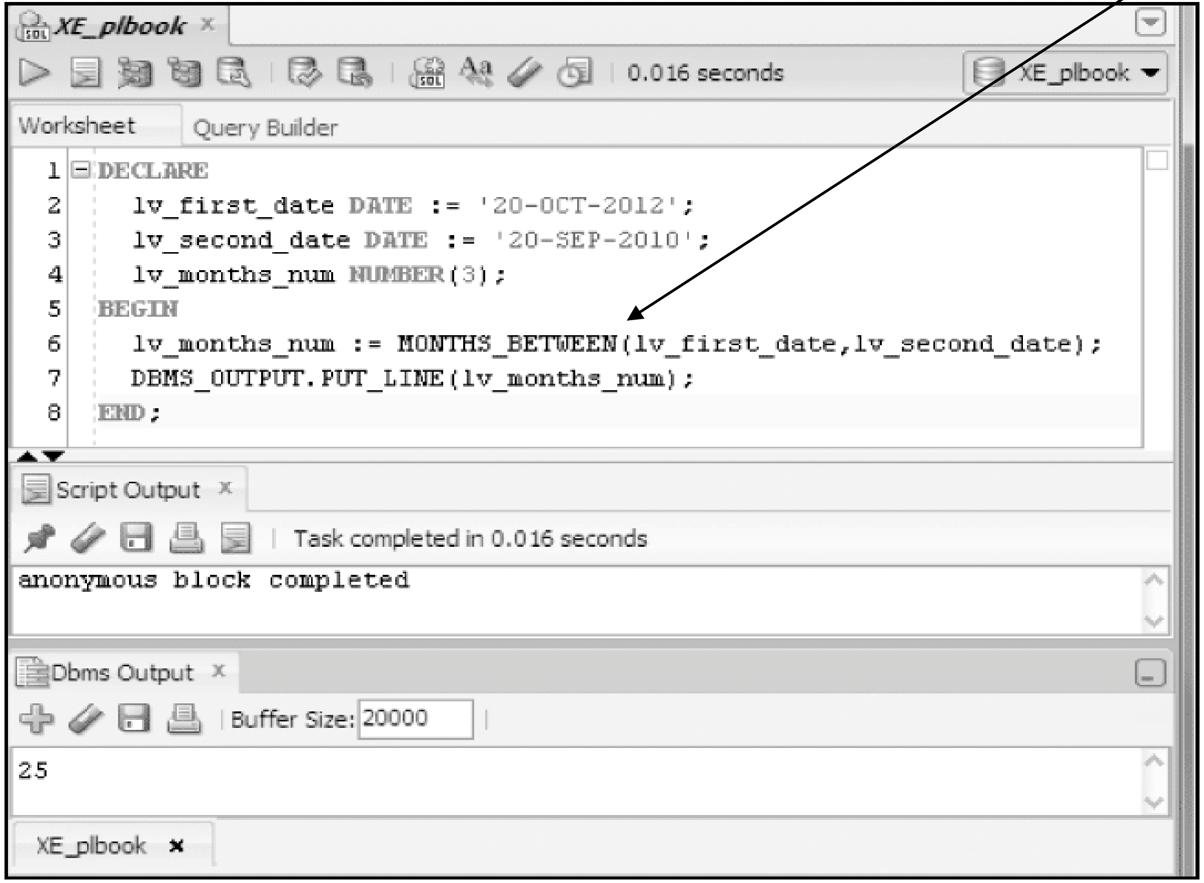
P
L
/
S
Q
L

multiplication

```
DECLARE
  lv_taxrate_num CONSTANT NUMBER(2,2) := .06;
  lv_total_num NUMBER(6,2) := 50;
  lv_taxamt_num NUMBER(4,2);
BEGIN
  lv_taxamt_num := lv_total_num * lv_taxrate_num;
  DBMS_OUTPUT.PUT_LINE(lv_taxamt_num);
END;
/
```

Using SQL Functions

•SQL functions such as MONTHS_BETWEEN can be used within PL/SQL statements



```
DECLARE
  lv_first_date DATE := '20-OCT-2012';
  lv_second_date DATE := '20-SEP-2010';
  lv_months_num NUMBER(3);
BEGIN
  lv_months_num := MONTHS_BETWEEN(lv_first_date,lv_second_date);
  DBMS_OUTPUT.PUT_LINE(lv_months_num);
END;
```

```
anonymous block completed
```

```
25
```

Decision Structures

P
L
/
S
Q
L

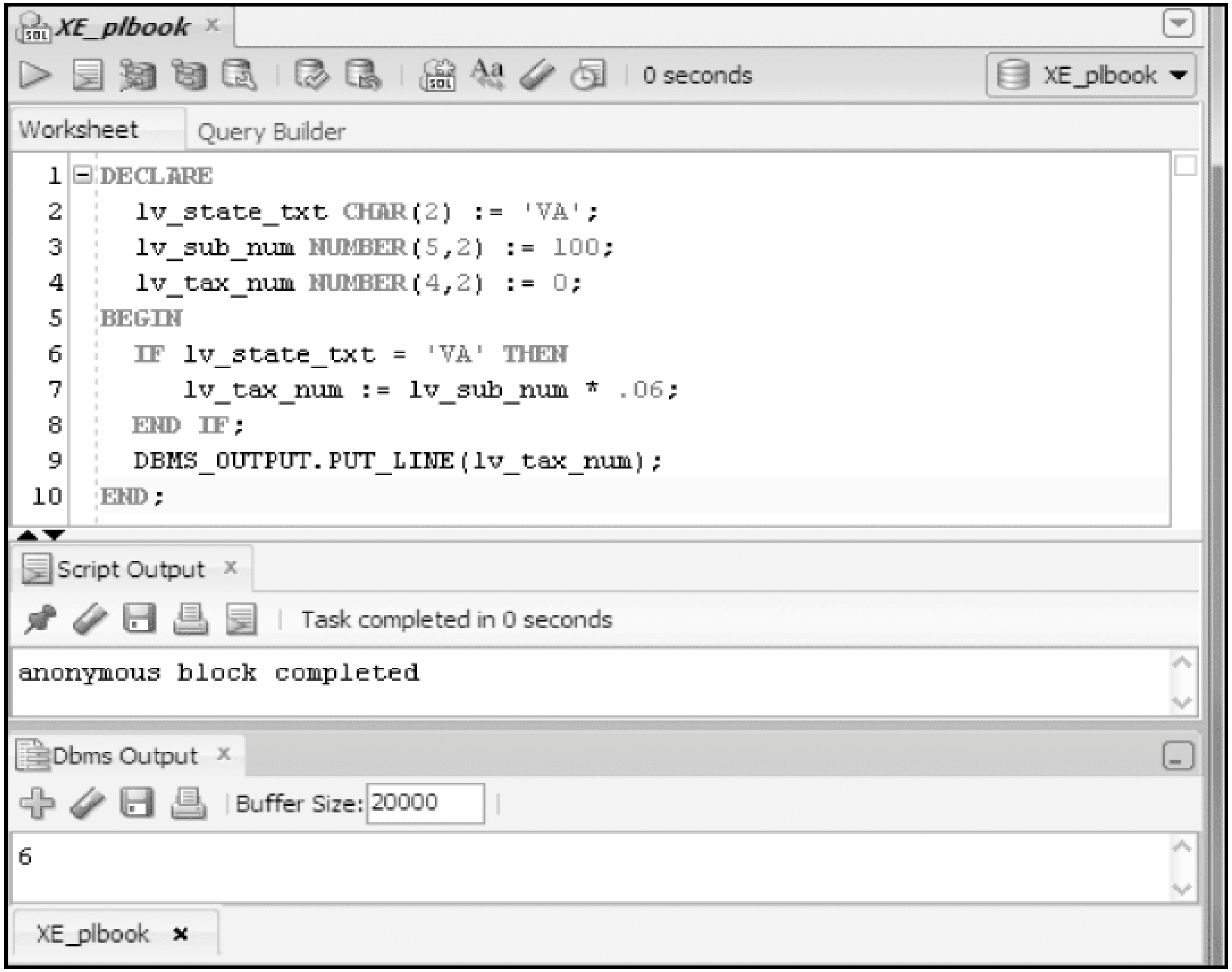
- Control which statements in a PL/SQL block will execute
- Enables conditions to be tested to determine the flow of statement execution
- Most programming languages provide IF and CASE statements to enable conditional processing

Decision Structures (continued)

- IF Statements
  - – Simple IF
  - – IF/THEN/ELSE
  - – IF/THEN/ELSIF/ELSE
- CASE Statements
  - – Basic CASE statement
  - – Searched CASE statement
  - – CASE expression

P
L
/
S
Q
L

Basic IF Statement

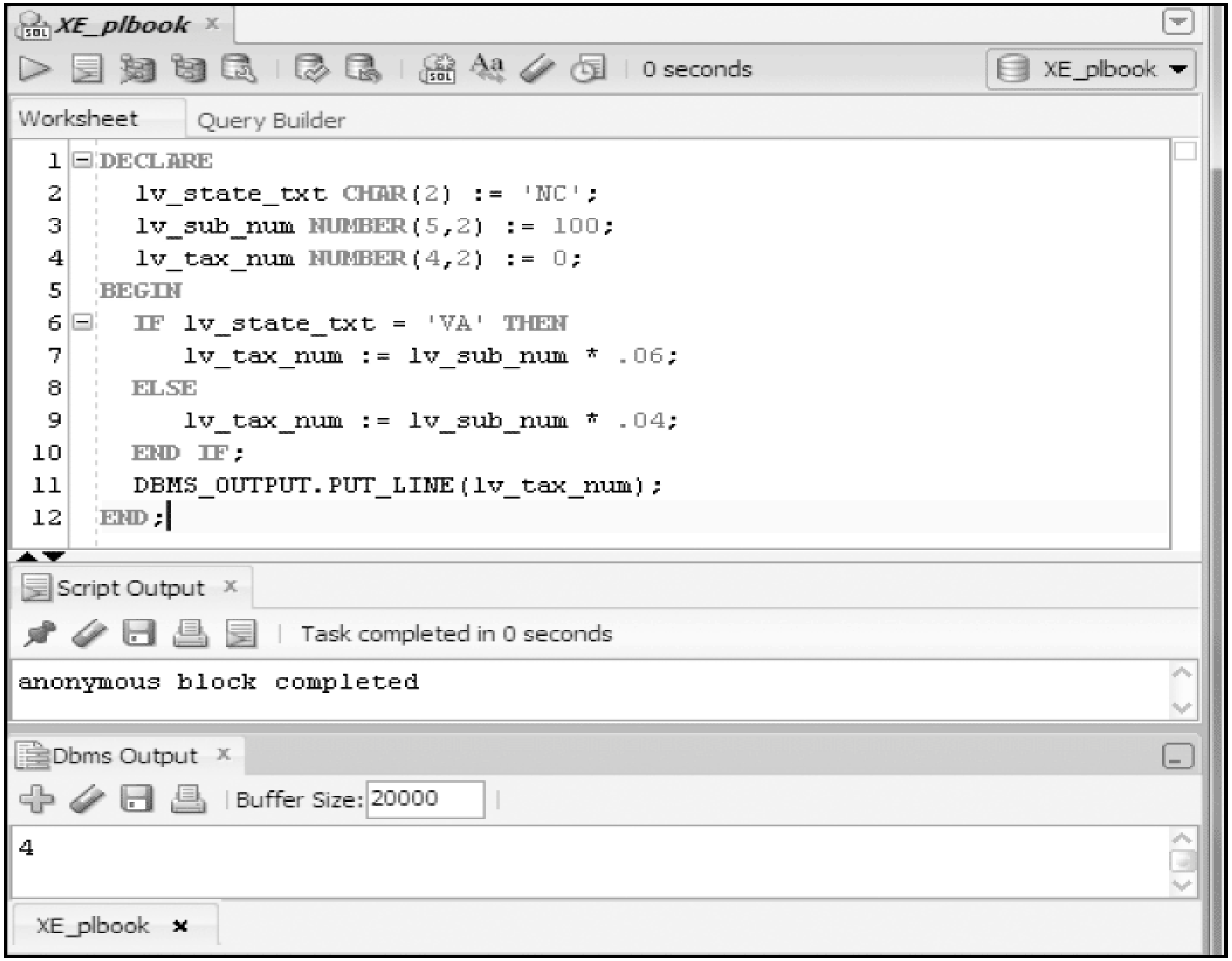


```
DECLARE
   lv_state_txt CHAR(2) := 'VA';
   lv_sub_num NUMBER(5,2) := 100;
   lv_tax_num NUMBER(4,2) := 0;
BEGIN
   IF lv_state_txt = 'VA' THEN
      lv_tax_num := lv_sub_num * .06;
   END IF;
   DBMS_OUTPUT.PUT_LINE(lv_tax_num);
END;
```

```
anonymous block completed
```

```
6
```

IF/THEN/ELSE



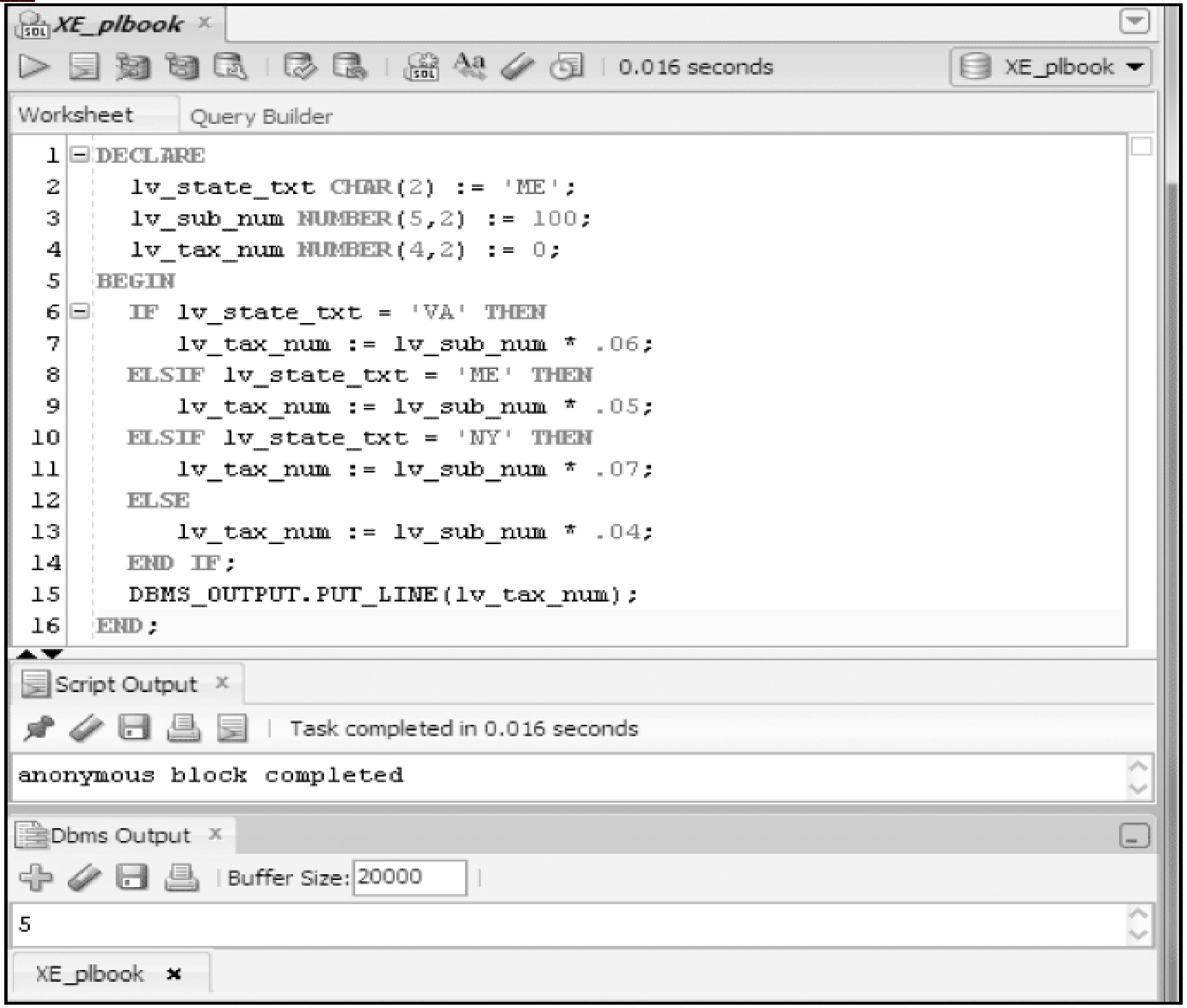
```
DECLARE
   lv_state_txt CHAR(2) := 'NC';
   lv_sub_num NUMBER(5,2) := 100;
   lv_tax_num NUMBER(4,2) := 0;
BEGIN
   IF lv_state_txt = 'VA' THEN
      lv_tax_num := lv_sub_num * .06;
   ELSE
      lv_tax_num := lv_sub_num * .04;
   END IF;
   DBMS_OUTPUT.PUT_LINE(lv_tax_num);
END;
```

```
anonymous block completed
```

```
4
```

IF/THEN/ELSIF/ELSE

P
L
/
S
Q
L



XE_plbook

0.016 seconds

Worksheet | Query Builder

```
DECLARE
   lv_state_txt CHAR(2) := 'ME';
   lv_sub_num NUMBER(5,2) := 100;
   lv_tax_num NUMBER(4,2) := 0;
BEGIN
   IF lv_state_txt = 'VA' THEN
      lv_tax_num := lv_sub_num * .06;
   ELSIF lv_state_txt = 'ME' THEN
      lv_tax_num := lv_sub_num * .05;
   ELSIF lv_state_txt = 'NY' THEN
      lv_tax_num := lv_sub_num * .07;
   ELSE
      lv_tax_num := lv_sub_num * .04;
   END IF;
   DBMS_OUTPUT.PUT_LINE(lv_tax_num);
END;
```

Script Output

Task completed in 0.016 seconds

```
anonymous block completed
```

Dbms Output

Buffer Size: 20000

```
5
```

XE_plbook

Nested IF

P
L
/
S
Q
L

```
IF lv_type_txt = 'E' THEN
   IF lv_price_num > 85 THEN      <-- Inner or nested IF begins
     lv_disc_num = .20;
   ELSIF lv_price_num > 45 THEN
     lv_disc_num = .15;
   ELSE
     lv_disc_num = .10;
   END IF;                          <-- Inner or nested IF ends
ELSIF lv_type_txt = 'C' THEN
   lv_disc_num = .05;
END IF;
```

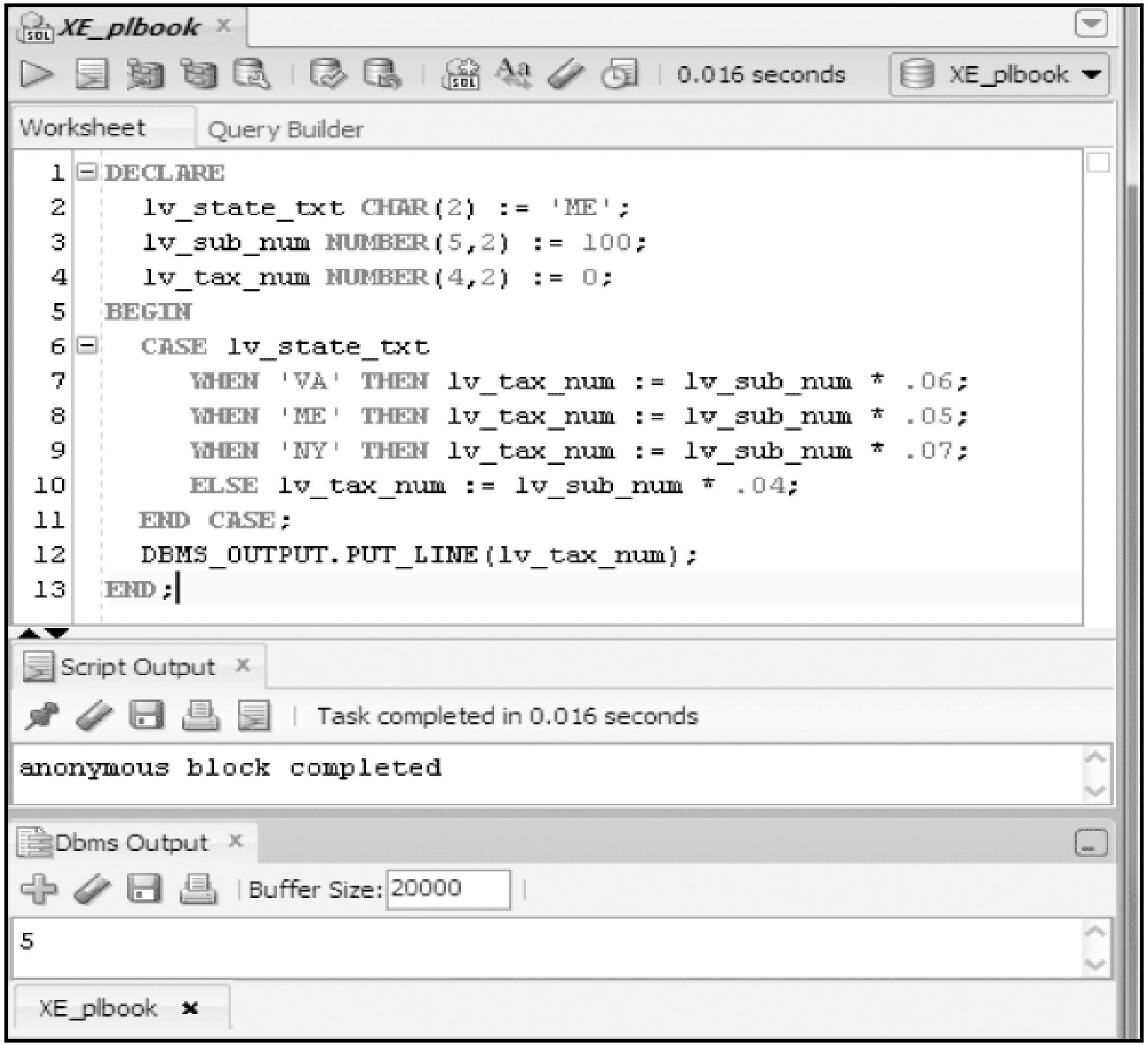
Logical Operators within IF

- Logical operators (AND, OR) enable multiple conditions to be checked

```
IF lv_state_txt = 'VA' OR lv_state_txt = 'PA' THEN
  lv_tax_num := lv_sub_num * .06;
ELSE
  lv_tax_num := lv_sub_num * .04;
END IF;
```

Basic CASE Statement

P
L
/
S
Q
L

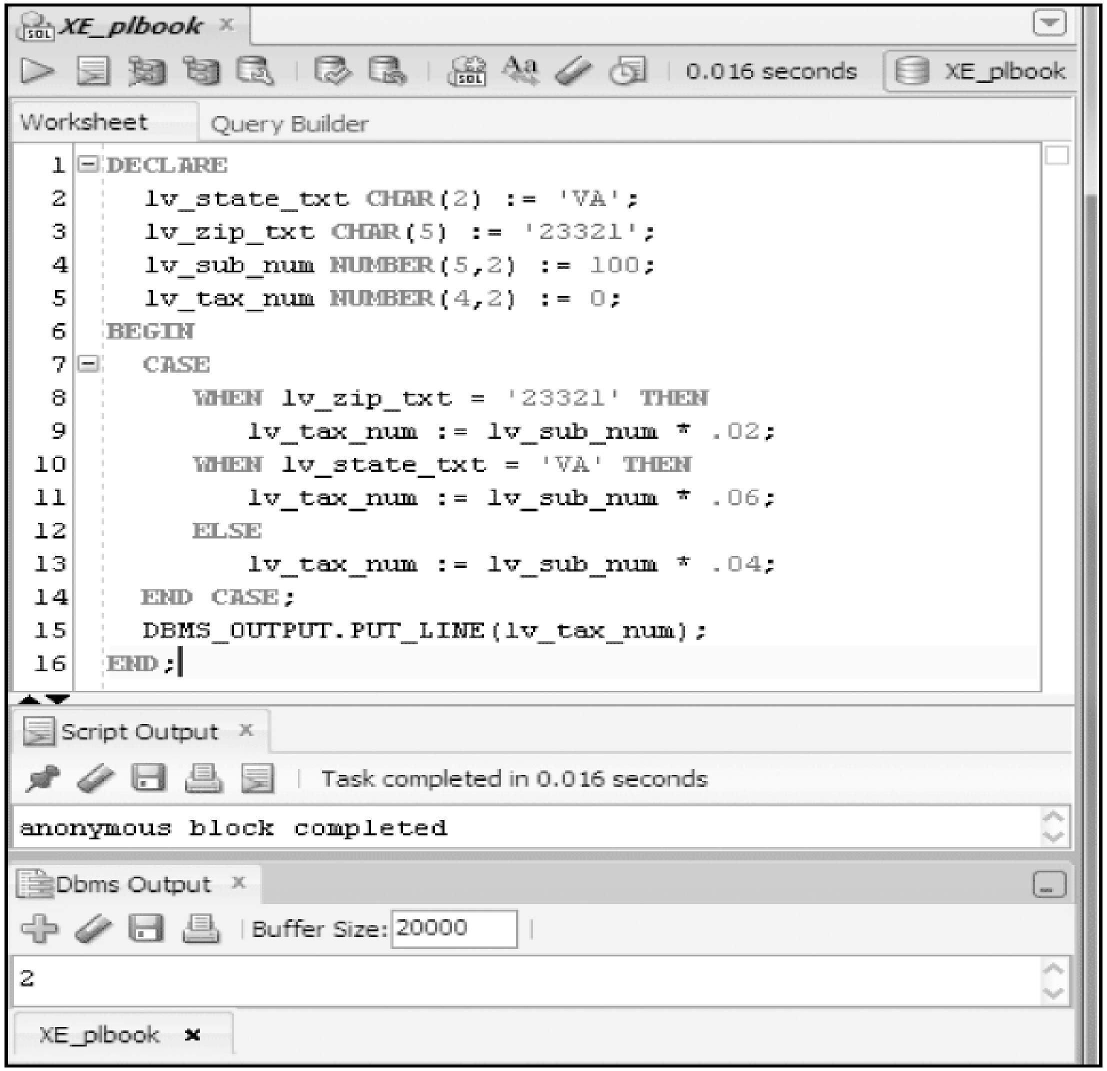


```
DECLARE
   lv_state_txt CHAR(2) := 'ME';
   lv_sub_num NUMBER(5,2) := 100;
   lv_tax_num NUMBER(4,2) := 0;
BEGIN
   CASE lv_state_txt
      WHEN 'VA' THEN lv_tax_num := lv_sub_num * .06;
      WHEN 'ME' THEN lv_tax_num := lv_sub_num * .05;
      WHEN 'NY' THEN lv_tax_num := lv_sub_num * .07;
      ELSE lv_tax_num := lv_sub_num * .04;
   END CASE;
   DBMS_OUTPUT.PUT_LINE(lv_tax_num);
END;
```

```
anonymous block completed
```

```
5
```

Searched CASE



```
DECLARE
   lv_state_txt CHAR(2) := 'VA';
   lv_zip_txt CHAR(5) := '23321';
   lv_sub_num NUMBER(5,2) := 100;
   lv_tax_num NUMBER(4,2) := 0;
BEGIN
   CASE
      WHEN lv_zip_txt = '23321' THEN
         lv_tax_num := lv_sub_num * .02;
      WHEN lv_state_txt = 'VA' THEN
         lv_tax_num := lv_sub_num * .06;
      ELSE
         lv_tax_num := lv_sub_num * .04;
   END CASE;
   DBMS_OUTPUT.PUT_LINE(lv_tax_num);
END;
```

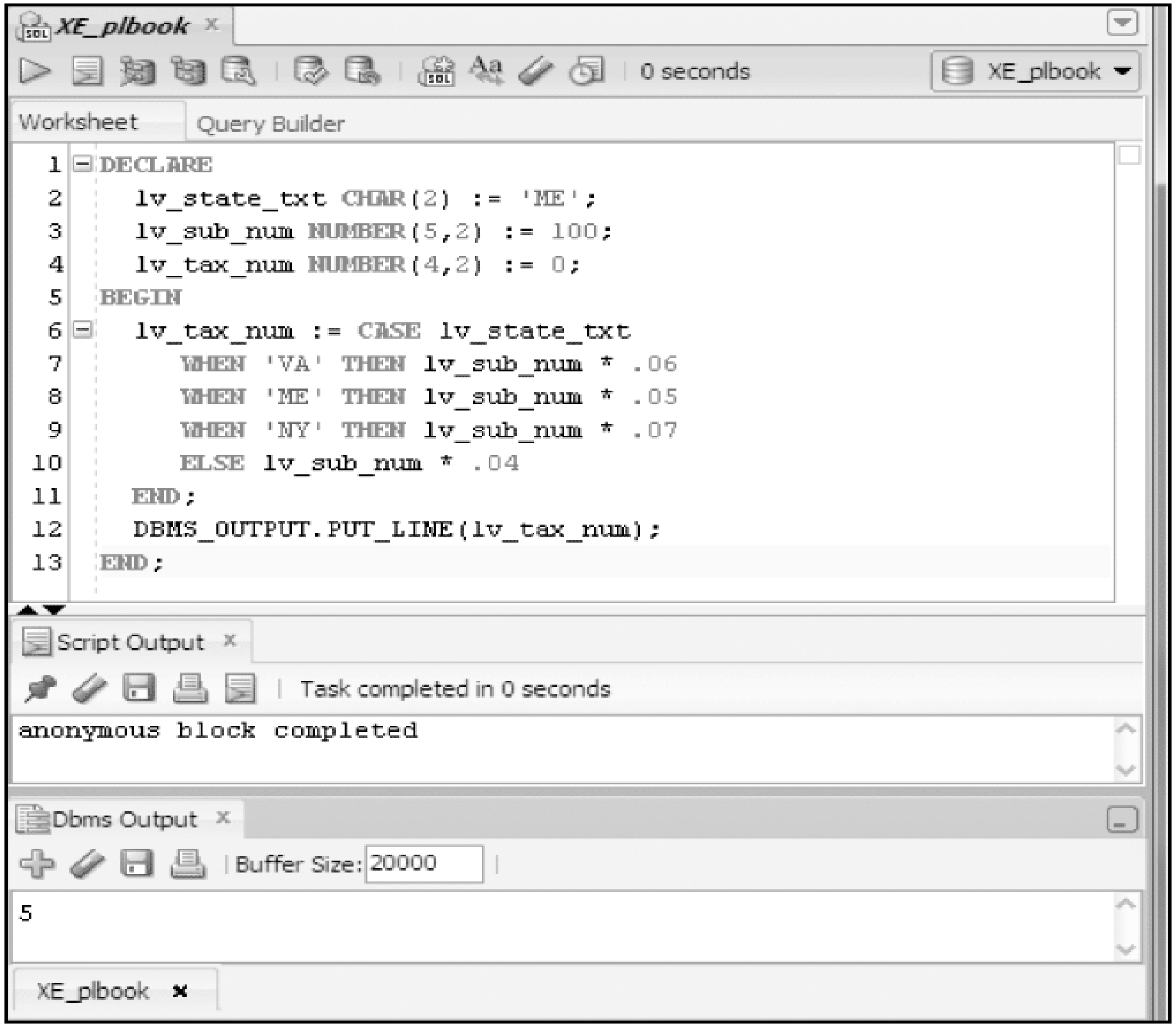
Task completed in 0.016 seconds

```
anonymous block completed
```

```
2
```

CASE Expression

P
L
/
S
Q
L



```
DECLARE
   lv_state_txt CHAR(2) := 'ME';
   lv_sub_num NUMBER(5,2) := 100;
   lv_tax_num NUMBER(4,2) := 0;
BEGIN
   lv_tax_num := CASE lv_state_txt
      WHEN 'VA' THEN lv_sub_num * .06
      WHEN 'ME' THEN lv_sub_num * .05
      WHEN 'NY' THEN lv_sub_num * .07
      ELSE lv_sub_num * .04
   END;
   DBMS_OUTPUT.PUT_LINE(lv_tax_num);
END;
```

```
anonymous block completed
```

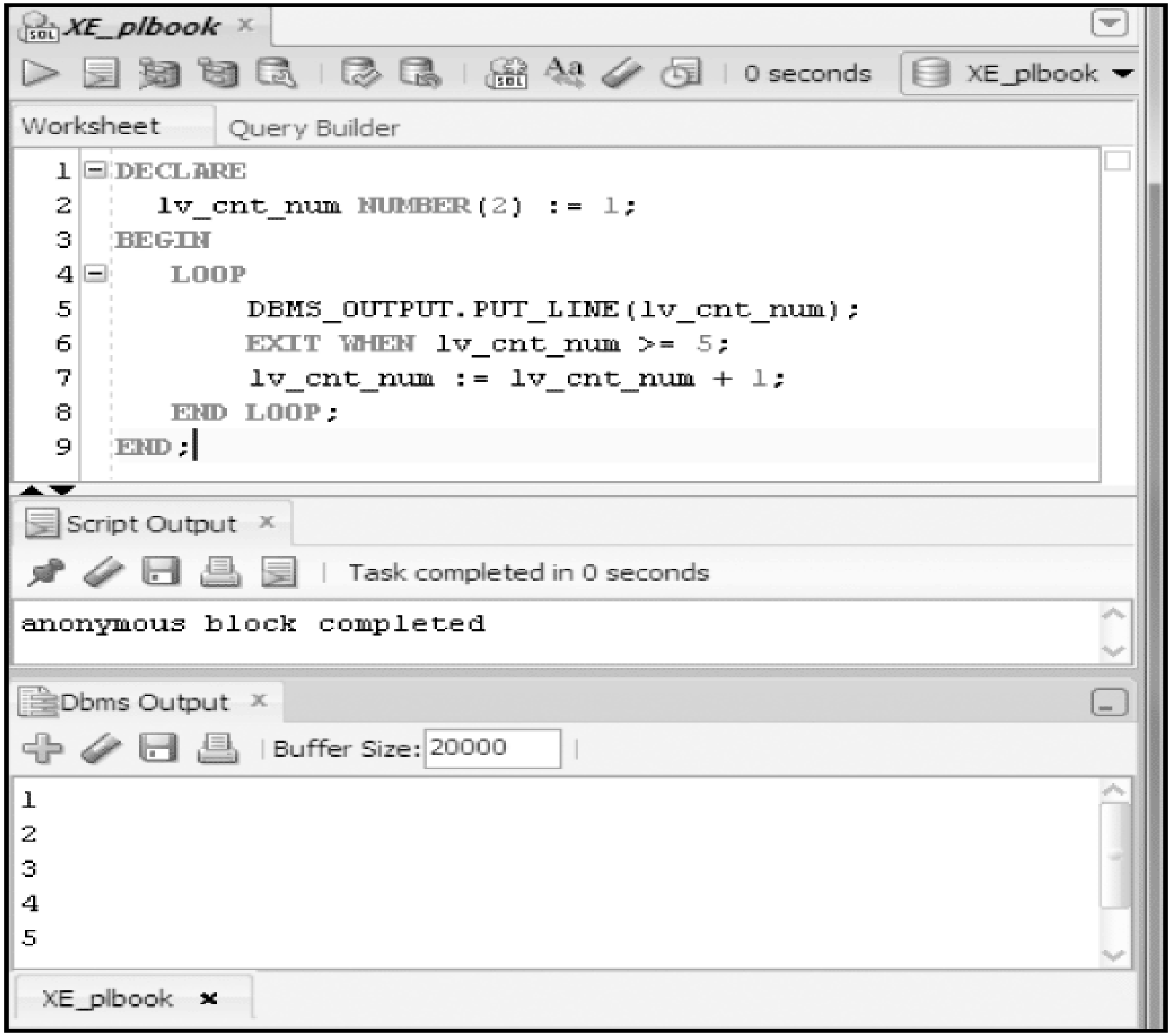
```
5
```

Looping

- Enables a statement or set of statements to be executed more than once
- A loop must provide instructions of when to end the looping, or an ‘infinite’ loop will be produced

Basic LOOP

P
L
/
S
Q
L

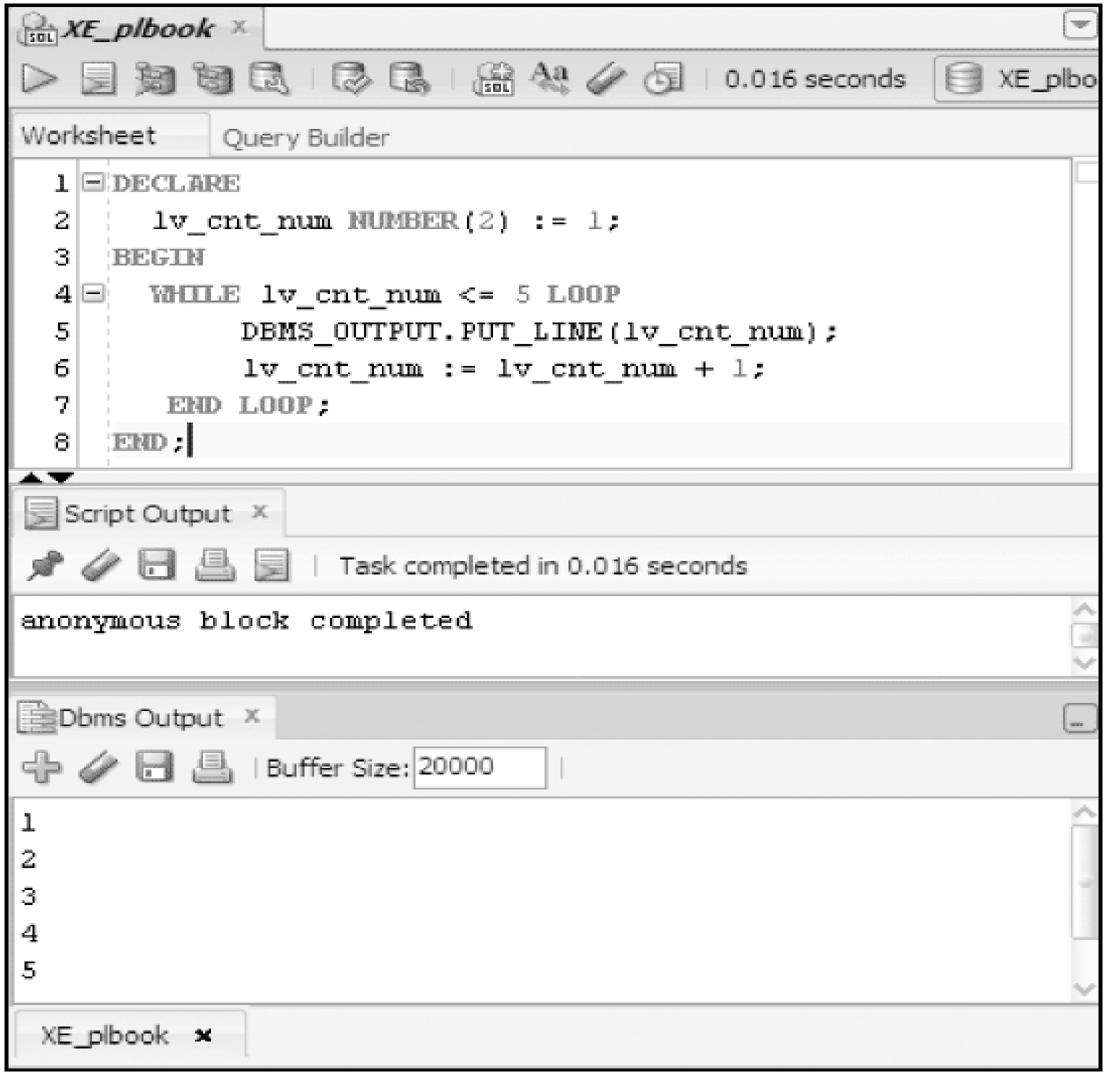


```
DECLARE
   lv_cnt_num NUMBER(2) := 1;
BEGIN
    LOOP
          DBMS_OUTPUT.PUT_LINE(lv_cnt_num);
          EXIT WHEN lv_cnt_num >= 5;
          lv_cnt_num := lv_cnt_num + 1;
    END LOOP;
END;
```

```
anonymous block completed
```

```
1
2
3
4
5
```

WHILE Loop



XE_plbook

0.016 seconds

Worksheet Query Builder

```
DECLARE
  lv_cnt_num NUMBER(2) := 1;
BEGIN
  WHILE lv_cnt_num <= 5 LOOP
       DBMS_OUTPUT.PUT_LINE(lv_cnt_num);
       lv_cnt_num := lv_cnt_num + 1;
   END LOOP;
END;
```

Script Output

Task completed in 0.016 seconds

```
anonymous block completed
```

Dbms Output

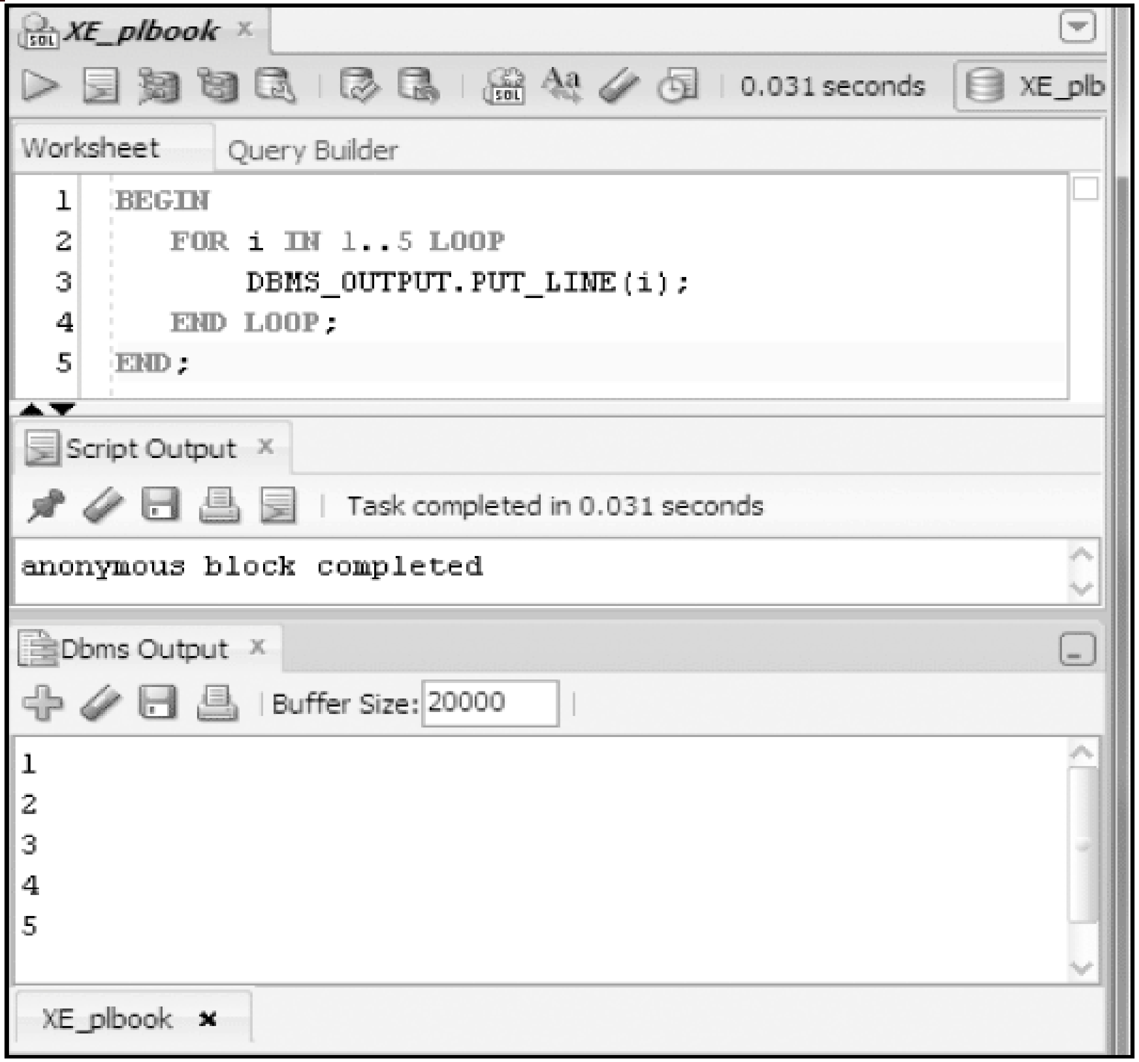
Buffer Size: 20000

```
1
2
3
4
5
```

XE_plbook

P
L
/
S
Q
L

FOR Loop



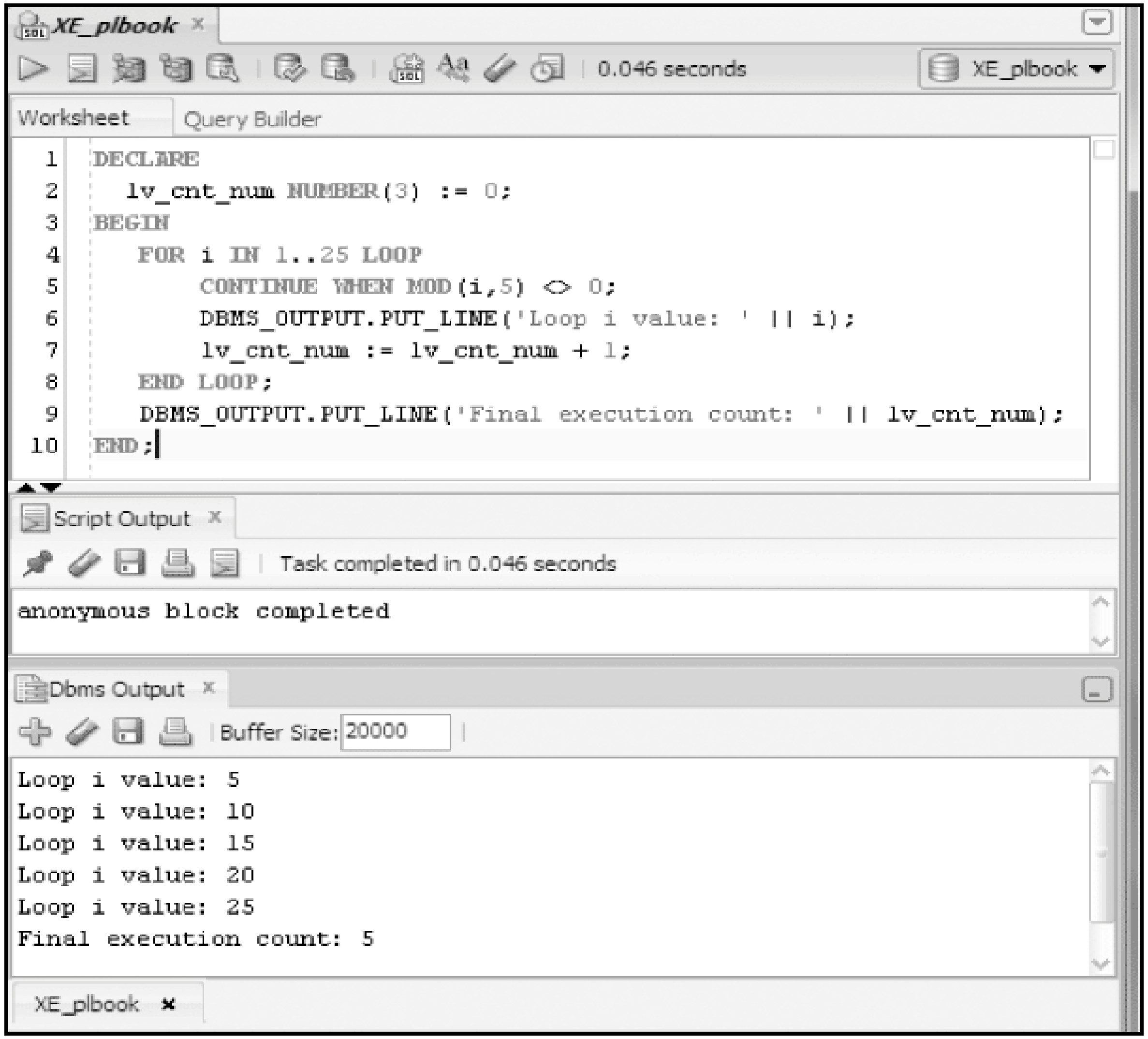
```
BEGIN
   FOR i IN 1..5 LOOP
       DBMS_OUTPUT.PUT_LINE(i);
   END LOOP;
END;
```

```
anonymous block completed
```

```
1
2
3
4
5
```

P
L
/
S
Q
L

CONTINUE Statement



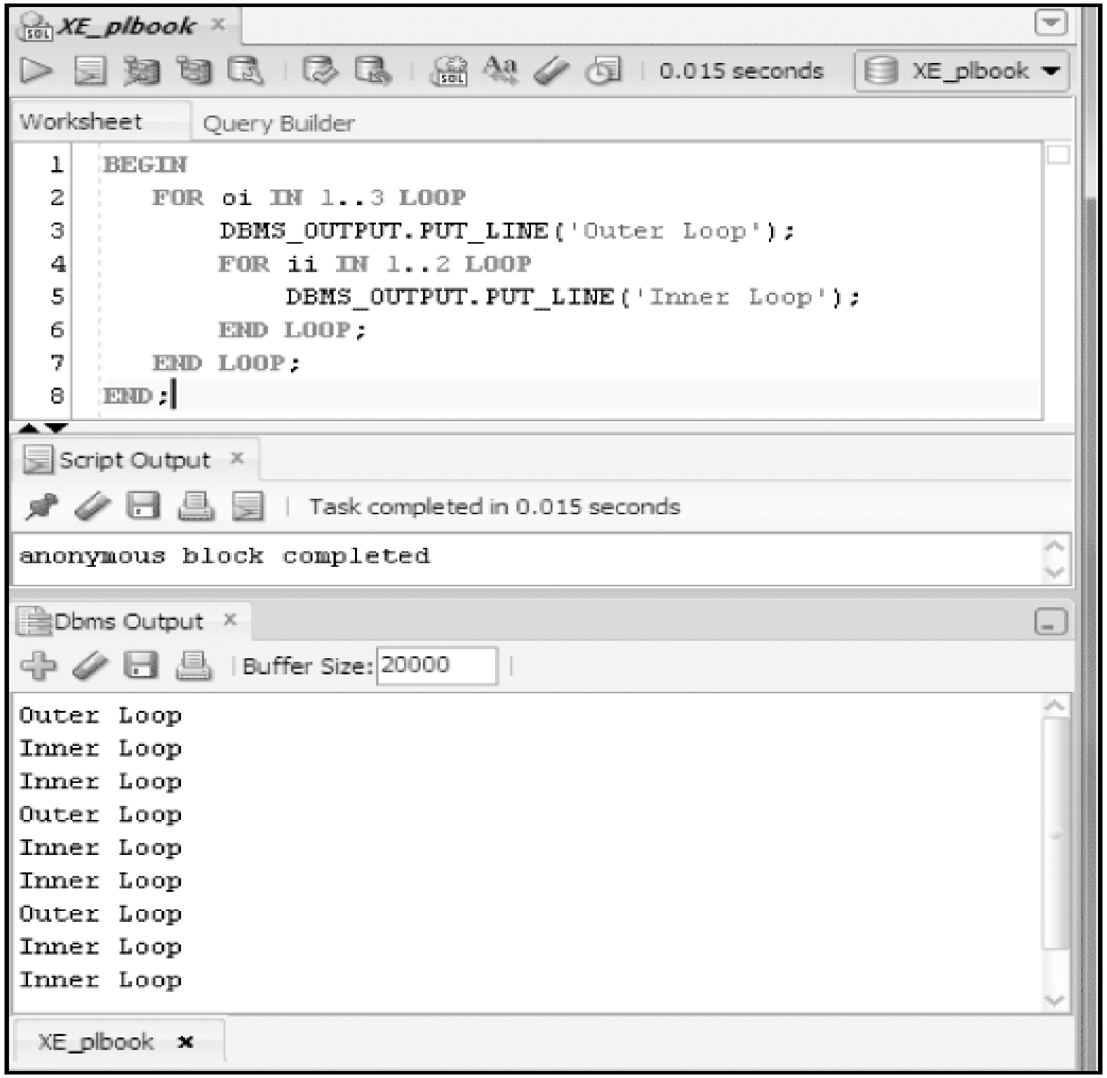
```
DECLARE
  lv_cnt_num NUMBER(3) := 0;
BEGIN
   FOR i IN 1..25 LOOP
       CONTINUE WHEN MOD(i,5) <> 0;
       DBMS_OUTPUT.PUT_LINE('Loop i value: ' || i);
       lv_cnt_num := lv_cnt_num + 1;
   END LOOP;
   DBMS_OUTPUT.PUT_LINE('Final execution count: ' || lv_cnt_num);
END;
```

```
anonymous block completed
```

```
Loop i value: 5
Loop i value: 10
Loop i value: 15
Loop i value: 20
Loop i value: 25
Final execution count: 5
```

P
L
/
S
Q
L

Nested Loops



XE_plbook

0.015 seconds

Worksheet | Query Builder

```
BEGIN
   FOR oi IN 1..3 LOOP
       DBMS_OUTPUT.PUT_LINE('Outer Loop');
       FOR ii IN 1..2 LOOP
           DBMS_OUTPUT.PUT_LINE('Inner Loop');
       END LOOP;
   END LOOP;
END;
```

Script Output

Task completed in 0.015 seconds

```
anonymous block completed
```

Dbms Output

Buffer Size: 20000

```
Outer Loop
Inner Loop
Inner Loop
Outer Loop
Inner Loop
Inner Loop
Outer Loop
Inner Loop
Inner Loop
```

Summary

- A flowchart assists in laying out processing logic
- A PL/SQL block contains a DECLARE, BEGIN, EXCEPTION, and END sections
- Variables to hold values are declared
- Scalar variables hold a single data value
- Scalar variables can hold string values, numbers, dates, and Boolean values
- DBMS_OUTPUT.PUT_LINE is used to display values

Summary (continued)

- IF statement structure is IF/THEN/ELSIF/ELSE
- CASE statements provide decision processing similar to IF statements
- Looping structures include: basic, WHILE, and FOR
- Host or bind variables can be used to interact with the application environment

P
L
/
S
Q
L

Oracle:
PL/SQL Programming

Chapter 3

Handling Data in PL/SQL Blocks

Chapter Objectives

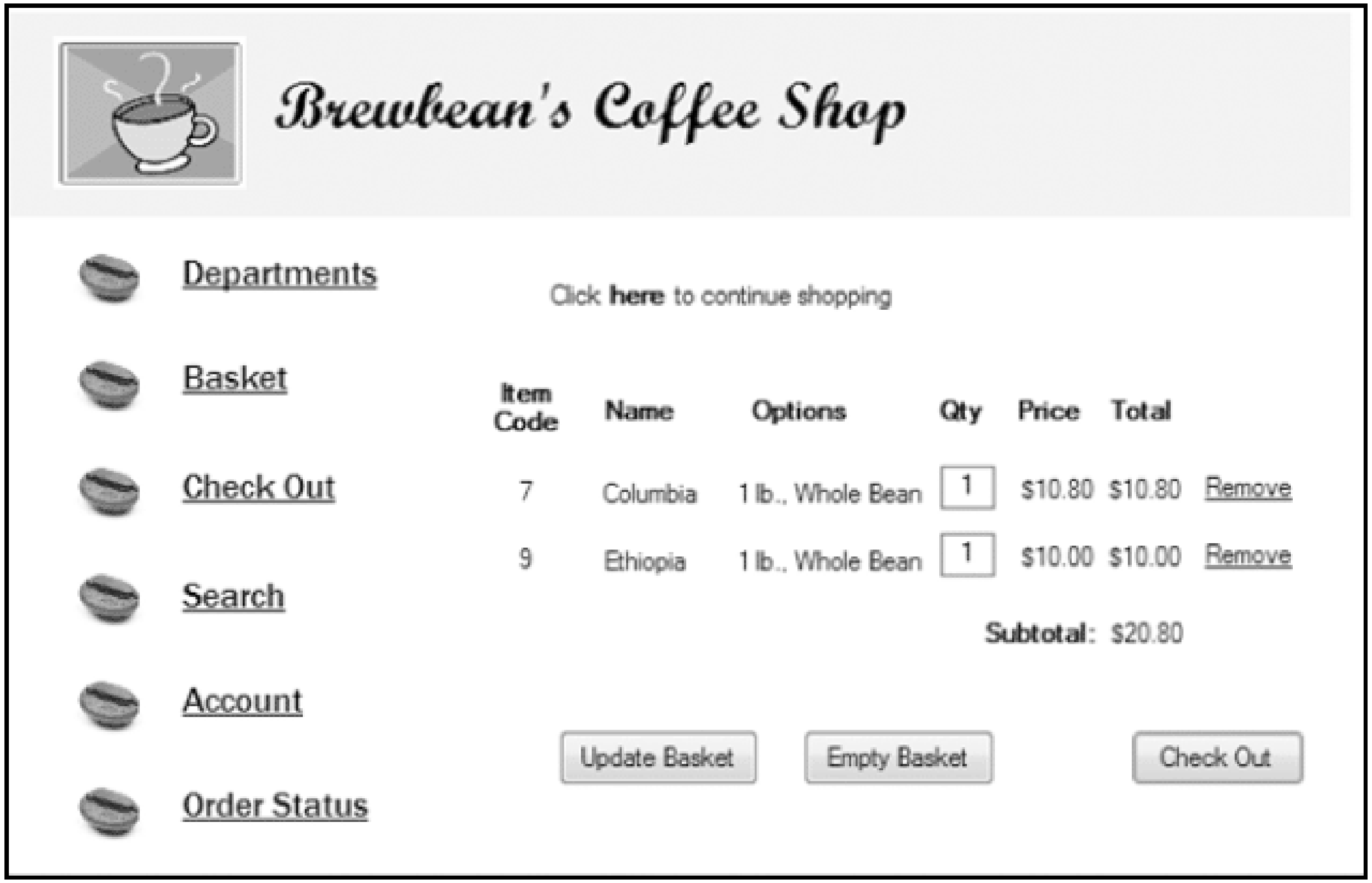
- After completing this lesson, you should be able to understand:
  - SQL queries in PL/SQL
  - The %TYPE attribute
  - Expanding block processing to include queries and control structures
  - Embedding DML statements in PL/SQL

Chapter Objectives (continued)

- After completing this lesson, you should be able to understand (continued):
  - Using record variables
  - Creating collections
  - Bulk processing basics
  - GOTO statement

Brewbean’s Challenge

- Consider actions needed upon check out



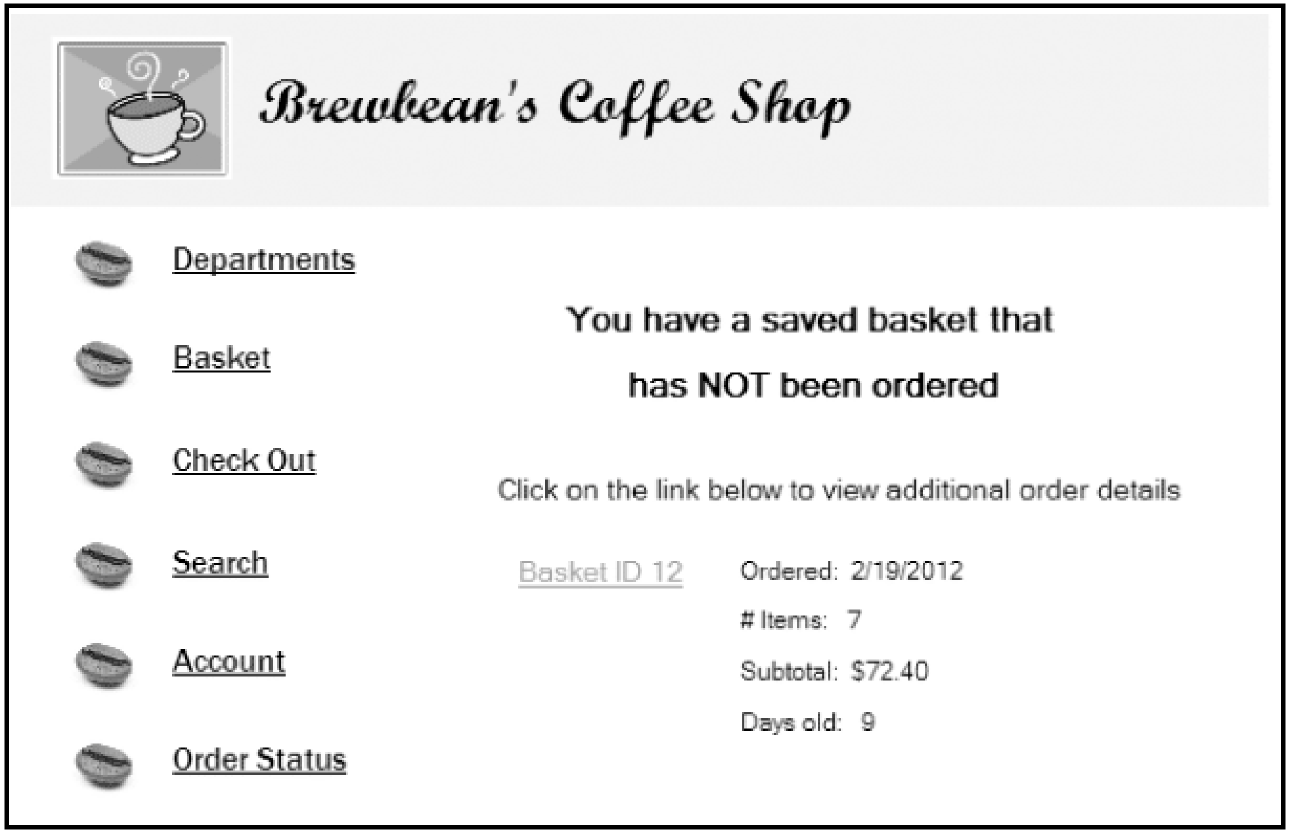
Click **here** to continue shopping

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

Include SQL within a Block

- Data query needs to identify if the customer has a saved basket



You have a saved basket that has NOT been ordered

Click on the link below to view additional order details

Basket ID 12

Ordered: 2/19/2012
Items: 7
Subtotal: $72.40
Days old: 9

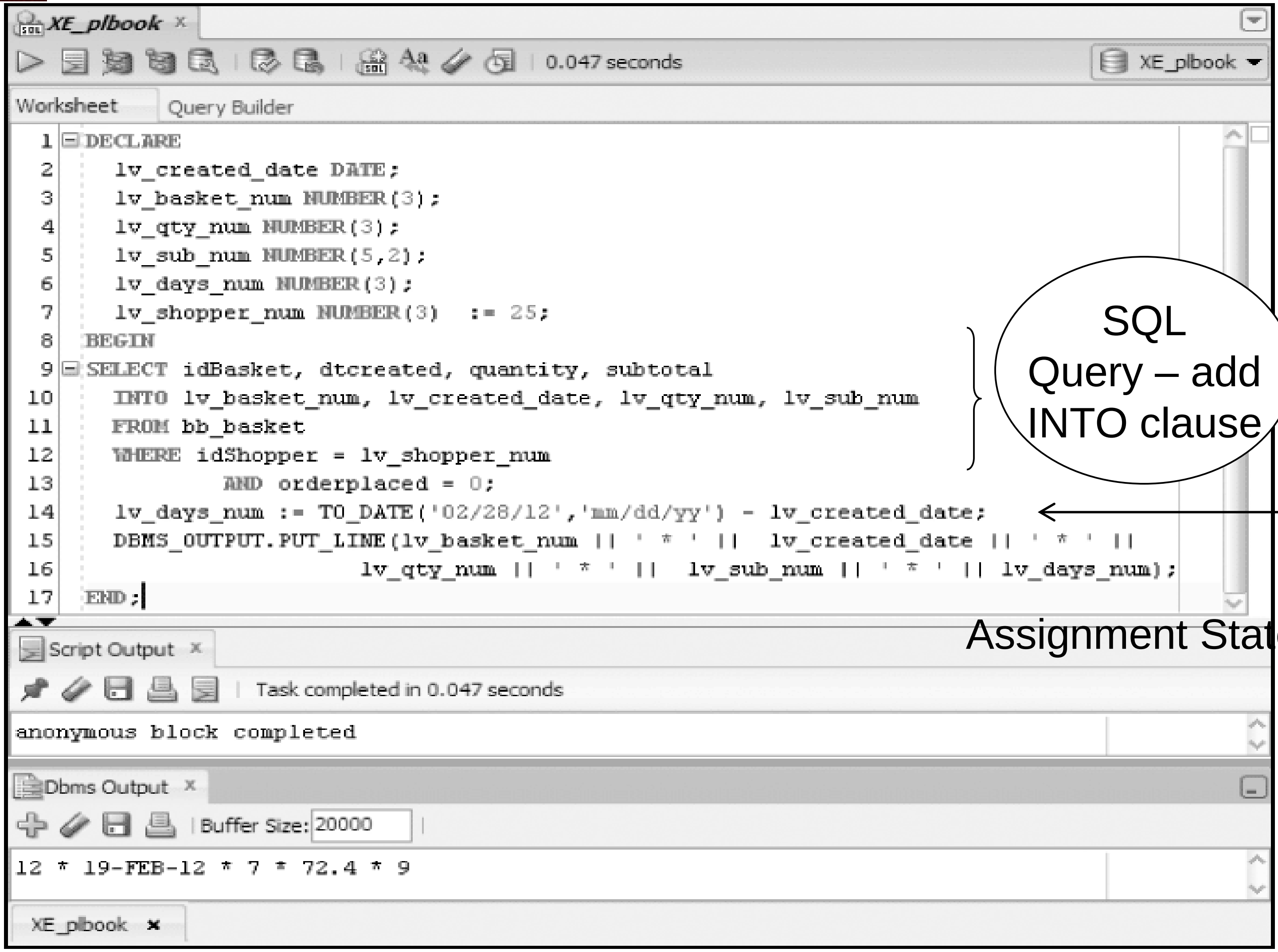
Include SQL within a Block (continued)

P
L
/
S
Q
L

- SQL statements can be embedded into the executable area of a PL/SQL block
- SELECT statements are embedded to query needed data
- An INTO clause is added to a SELECT statement to move data retrieved into variables

Include SQL within a Block (continued)

P
L
/
S
Q
L



```
DECLARE
   lv_created_date DATE;
   lv_basket_num NUMBER(3);
   lv_qty_num NUMBER(3);
   lv_sub_num NUMBER(5,2);
   lv_days_num NUMBER(3);
   lv_shopper_num NUMBER(3)  := 25;
BEGIN
SELECT idBasket, dtcreated, quantity, subtotal
   INTO lv_basket_num, lv_created_date, lv_qty_num, lv_sub_num
   FROM bb_basket
   WHERE idShopper = lv_shopper_num
           AND orderplaced = 0;
   lv_days_num := TO_DATE('02/28/12','mm/dd/yy') - lv_created_date;
   DBMS_OUTPUT.PUT_LINE(lv_basket_num || ' * ' ||  lv_created_date || ' * ' ||
                        lv_qty_num || ' * ' ||  lv_sub_num || ' * ' || lv_days_num);
END;
```

```
anonymous block completed
```

```
12 * 19-FEB-12 * 7 * 72.4 * 9
```

SQL Query – add INTO clause

Assignment Statement

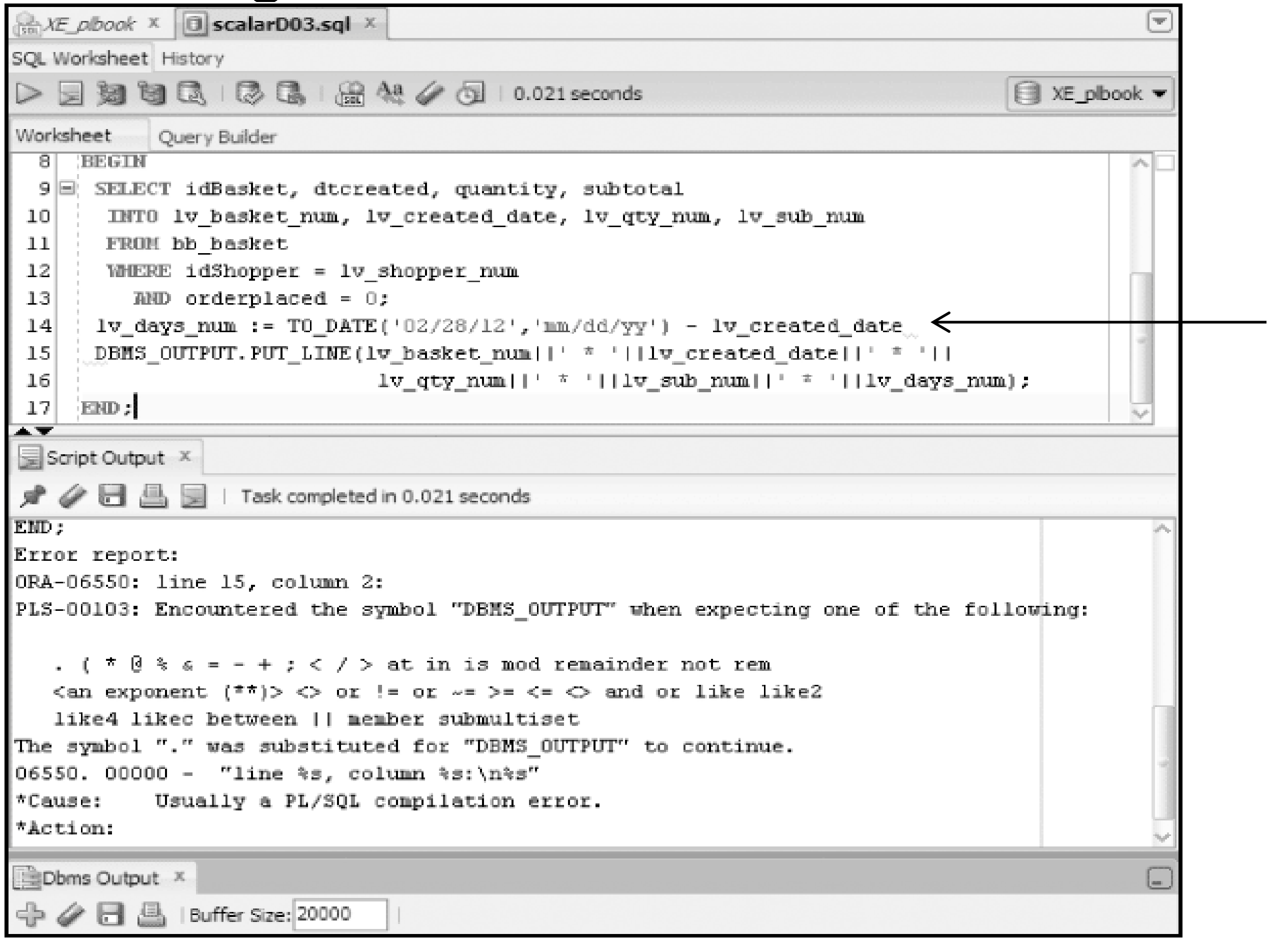
P
L
/
S
Q
L

Executing a Block with Errors

- Common Errors
  - Use = rather than :=
  - Not declaring a variable
  - Misspelling a variable name
  - Not ending a statement with ;
  - No data returned from a SELECT statement

Executing a Block with Errors (continued)

- Not closing a statement with ;



```
BEGIN
 SELECT idBasket, dtcreated, quantity, subtotal
  INTO lv_basket_num, lv_created_date, lv_qty_num, lv_sub_num
  FROM bb_basket
  WHERE idShopper = lv_shopper_num
    AND orderplaced = 0;
 lv_days_num := TO_DATE('02/28/12','mm/dd/yy') - lv_created_date
 DBMS_OUTPUT.PUT_LINE(lv_basket_num||' * '||lv_created_date||' * '||
                      lv_qty_num||' * '||lv_sub_num||' * '||lv_days_num);
END;
```

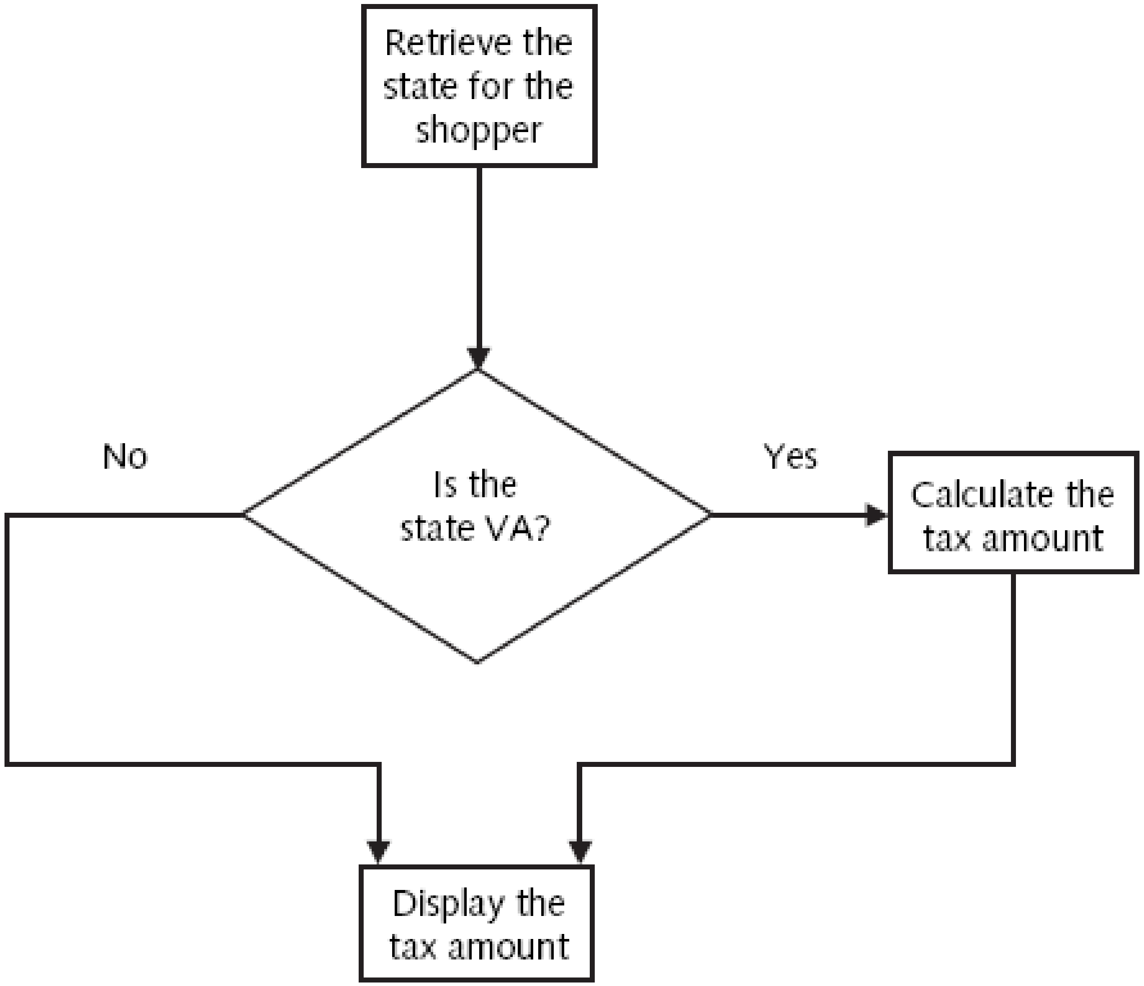
```
END;
Error report:
ORA-06550: line 15, column 2:
PLS-00103: Encountered the symbol "DBMS_OUTPUT" when expecting one of the following:

   . ( * @ % & = - + ; < / > at in is mod remainder not rem
   <an exponent (**)> <> or != or ~= >= <= <> and or like like2
   like4 likec between || member submultiset
The symbol "." was substituted for "DBMS_OUTPUT" to continue.
06550. 00000 -  "line %s, column %s:\n%s"
*Cause:    Usually a PL/SQL compilation error.
*Action:
```

%TYPE Attribute

- Use in variable declaration to provide data type based on a table column
- Ideal for declaring variables that will hold data from the database
- Minimizes maintenance by avoiding program changes to reflect database column changes
- Called an anchored data type

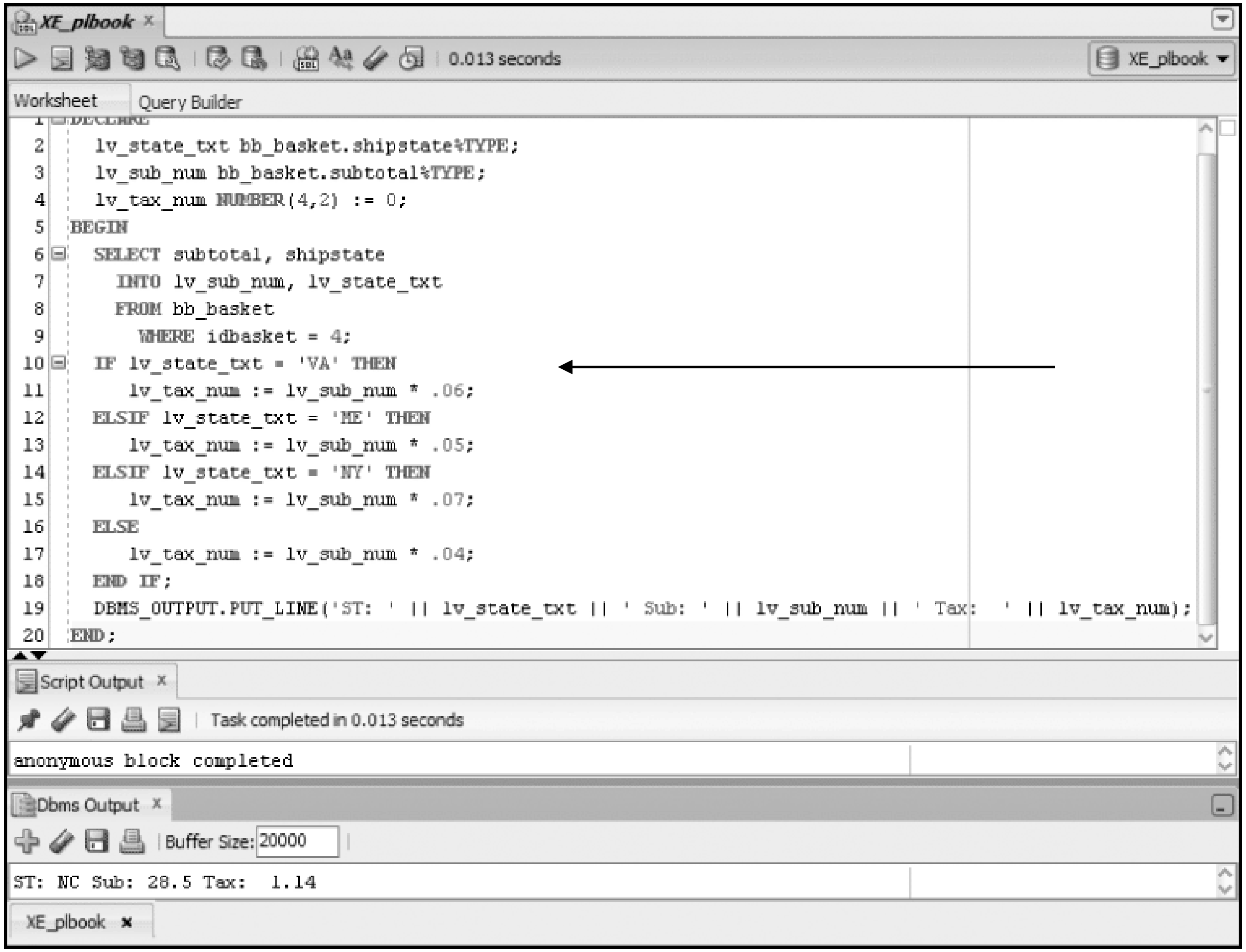
```
lv_basket_num bb_basket.idBasket%TYPE;
```

Data Retrieval with Decision Structures



P
L
/
S
Q
L

IF Statement Example



XE_plbook

0.013 seconds

Worksheet Query Builder

```
DECLARE
  lv_state_txt bb_basket.shipstate%TYPE;
  lv_sub_num bb_basket.subtotal%TYPE;
  lv_tax_num NUMBER(4,2) := 0;
BEGIN
  SELECT subtotal, shipstate
    INTO lv_sub_num, lv_state_txt
    FROM bb_basket
     WHERE idbasket = 4;
  IF lv_state_txt = 'VA' THEN
     lv_tax_num := lv_sub_num * .06;
  ELSIF lv_state_txt = 'ME' THEN
     lv_tax_num := lv_sub_num * .05;
  ELSIF lv_state_txt = 'NY' THEN
     lv_tax_num := lv_sub_num * .07;
  ELSE
     lv_tax_num := lv_sub_num * .04;
  END IF;
  DBMS_OUTPUT.PUT_LINE('ST: ' || lv_state_txt || ' Sub: ' || lv_sub_num || ' Tax:  ' || lv_tax_num);
END;
```

Script Output

Task completed in 0.013 seconds

```
anonymous block completed
```

Dbms Output

Buffer Size: 20000

```
ST: NC Sub: 28.5 Tax:  1.14
```

Including DML

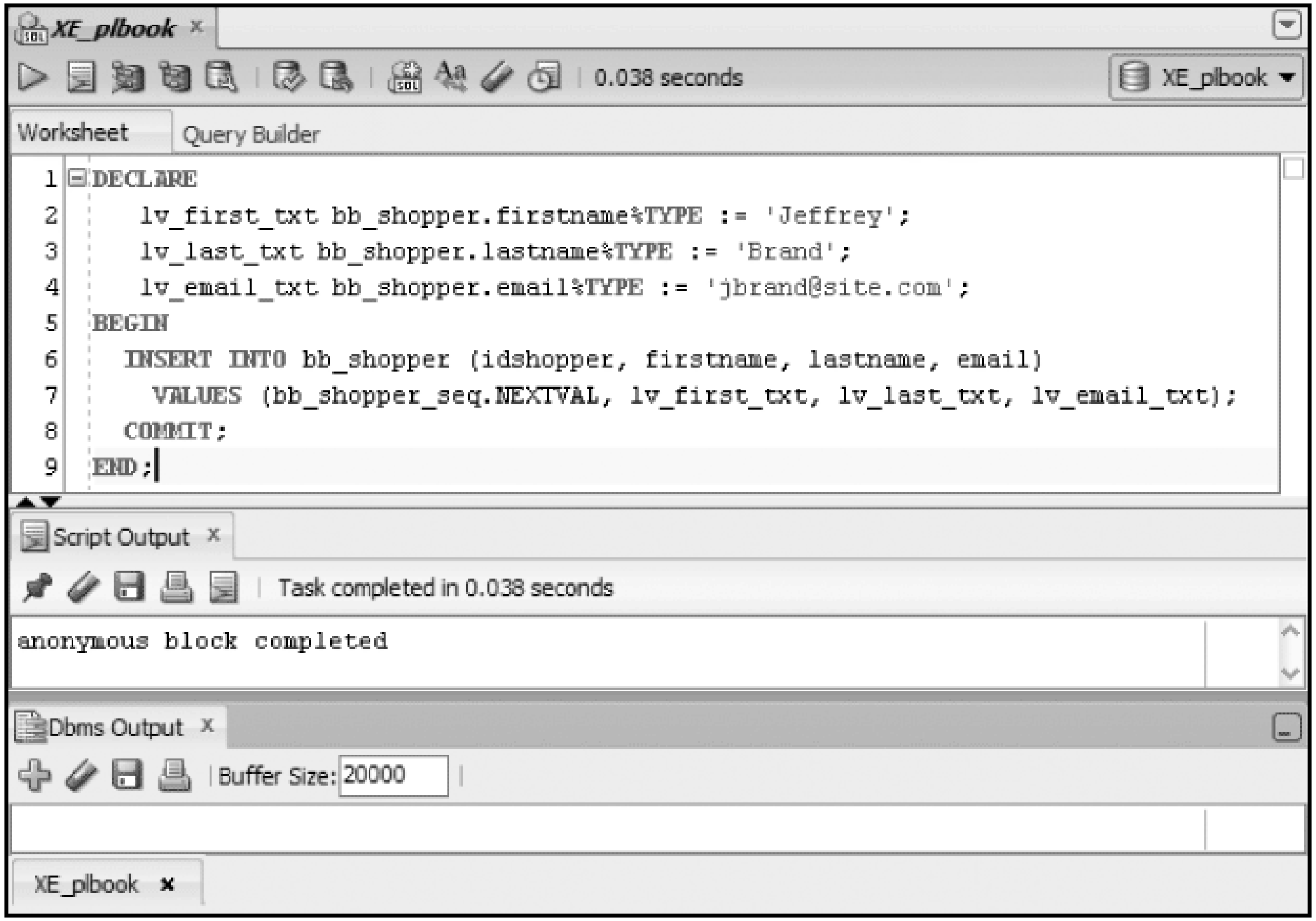
P
L
/
S
Q
L

- DML statements can be embedded into PL/SQL blocks to accomplish data changes
- DML includes INSERT, UPDATE, and DELETE statements

P
L
/
S
Q
L

Including DML (continued)

- Add a new shopper - INSERT



```
DECLARE
   lv_first_txt bb_shopper.firstname%TYPE := 'Jeffrey';
   lv_last_txt bb_shopper.lastname%TYPE := 'Brand';
   lv_email_txt bb_shopper.email%TYPE := 'jbrand@site.com';
BEGIN
  INSERT INTO bb_shopper (idshopper, firstname, lastname, email)
    VALUES (bb_shopper_seq.NEXTVAL, lv_first_txt, lv_last_txt, lv_email_txt);
  COMMIT;
END;
```

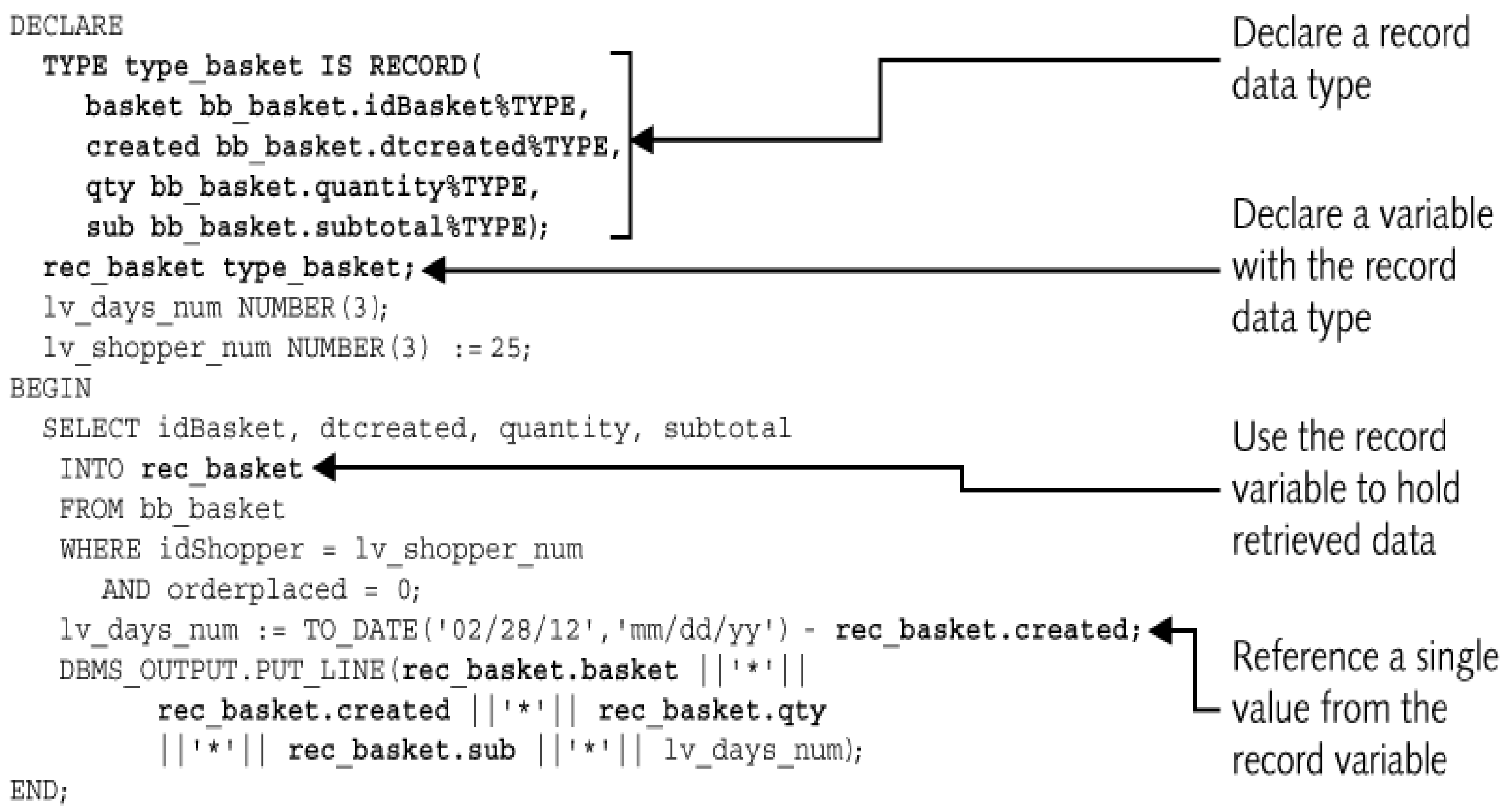
```
anonymous block completed
```

P
L
/
S
Q
L

Record variables

- Stores multiple values of different data types as one unit
- Record – can hold one row of data

Record Data Type


Declare a record data type
Declare a variable with the record data type
Use the record variable to hold retrieved data
Reference a single value from the record variable

```
DECLARE
  TYPE type_basket IS RECORD(
     basket bb_basket.idBasket%TYPE,
     created bb_basket.dtcreated%TYPE,
     qty bb_basket.quantity%TYPE,
     sub bb_basket.subtotal%TYPE);
  rec_basket type_basket;
  lv_days_num NUMBER(3);
  lv_shopper_num NUMBER(3) := 25;
BEGIN
  SELECT idBasket, dtcreated, quantity, subtotal
   INTO rec_basket
   FROM bb_basket
   WHERE idShopper = lv_shopper_num
      AND orderplaced = 0;
   lv_days_num := TO_DATE('02/28/12','mm/dd/yy') - rec_basket.created;
   DBMS_OUTPUT.PUT_LINE(rec_basket.basket ||'*'||
         rec_basket.created ||'*'|| rec_basket.qty
         ||'*'|| rec_basket.sub ||'*'|| lv_days_num);
END;
```

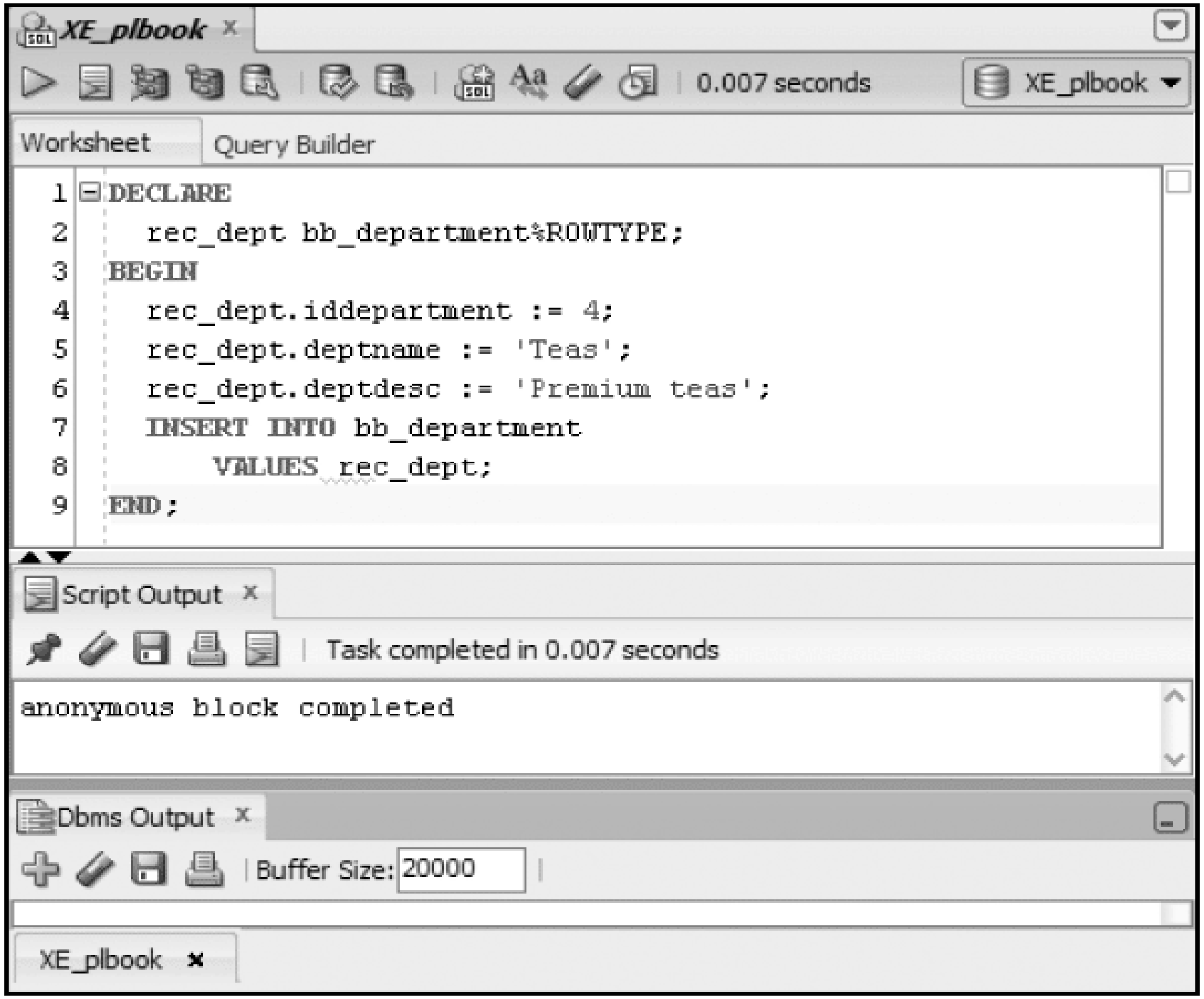
%ROWTYPE Attribute

- Create record structure based on table structure

```
DECLARE
 rec_shopper bb_shopper%ROWTYPE;
BEGIN
 SELECT *
  INTO rec_shopper
  FROM bb_shopper
  WHERE idshopper = 25;
 DBMS_OUTPUT.PUT_LINE(rec_shopper.lastname);
 DBMS_OUTPUT.PUT_LINE(rec_shopper.address);
 DBMS_OUTPUT.PUT_LINE(rec_shopper.email);
END;
```

INSERT Using Record

P
L
/
S
Q
L



```
DECLARE
  rec_dept bb_department%ROWTYPE;
BEGIN
  rec_dept.iddepartment := 4;
  rec_dept.deptname := 'Teas';
  rec_dept.deptdesc := 'Premium teas';
  INSERT INTO bb_department
      VALUES rec_dept;
END;
```

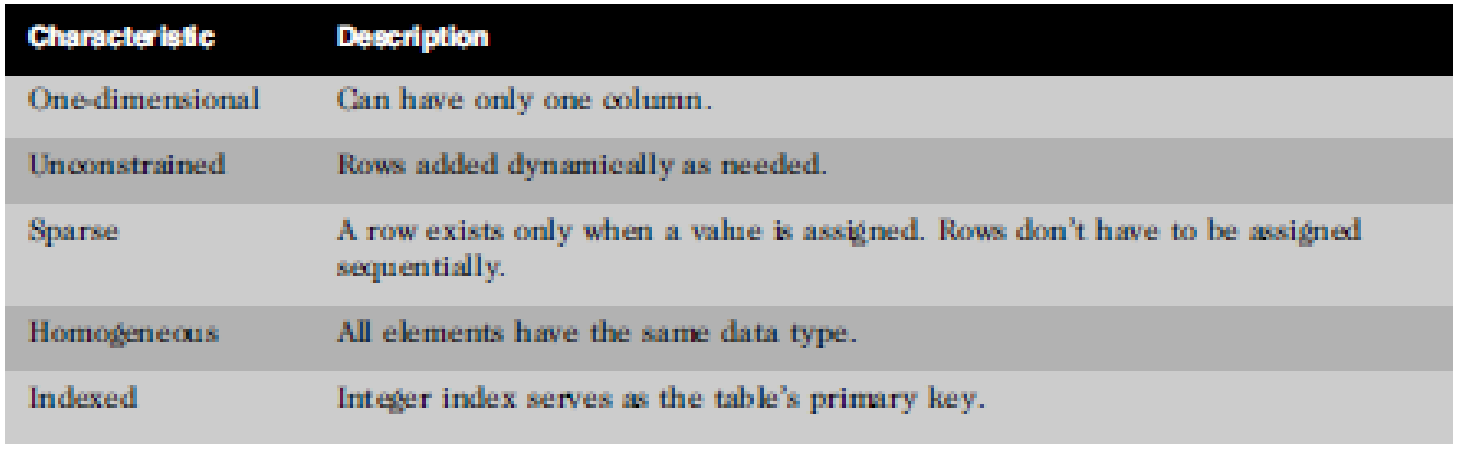
```
anonymous block completed
```

Collections

P
L
/
S
Q
L

- Store multiple values of the same data type
- Similar to arrays in other languages
- Associative Array– handle many rows of one field

TABLE 3-1 Associative Array Characteristics



Characteristic	Description
One-dimensional	Can have only one column.
Unconstrained	Rows added dynamically as needed.
Sparse	A row exists only when a value is assigned. Rows don't have to be assigned sequentially.
Homogeneous	All elements have the same data type.
Indexed	Integer index serves as the table's primary key.

Associative Array Attributes

P
L
/
S
Q
L

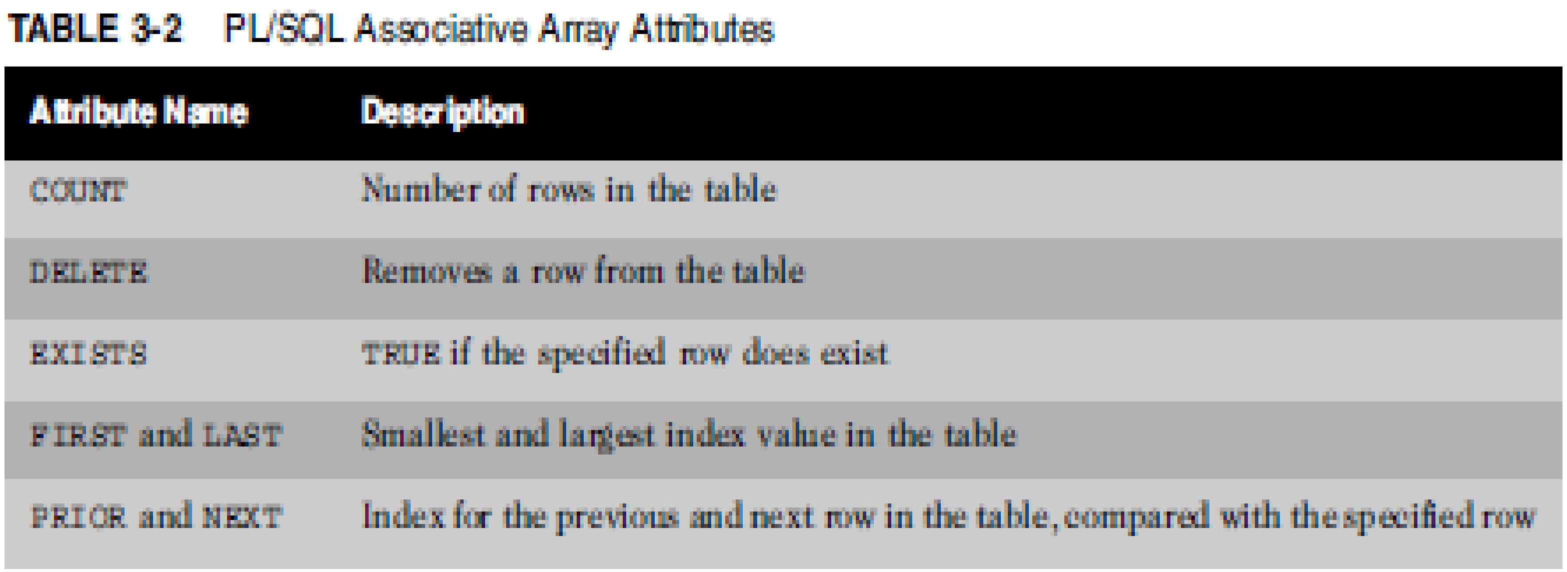
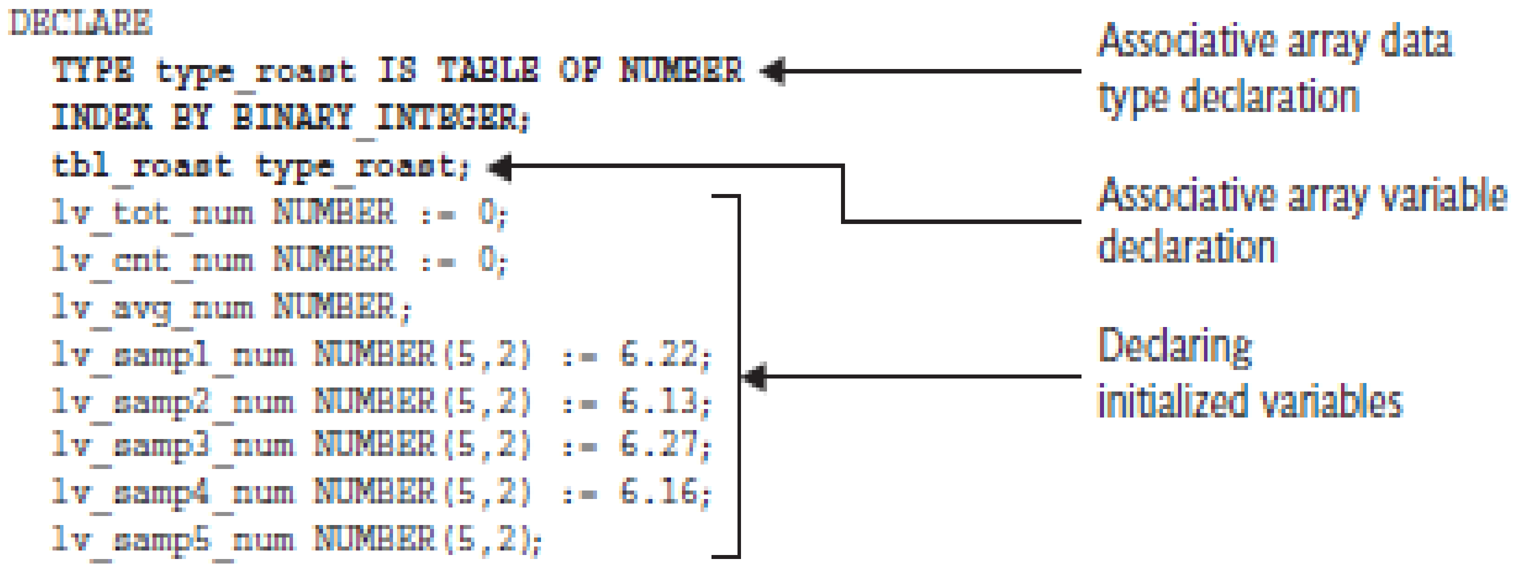


TABLE 3-2 PL/SQL Associative Array Attributes

Attribute Name	Description
`COUNT`	Number of rows in the table
`DELETE`	Removes a row from the table
`EXISTS`	`TRUE` if the specified row does exist
`FIRST` and `LAST`	Smallest and largest index value in the table
`PRIOR` and `NEXT`	Index for the previous and next row in the table, compared with the specified row

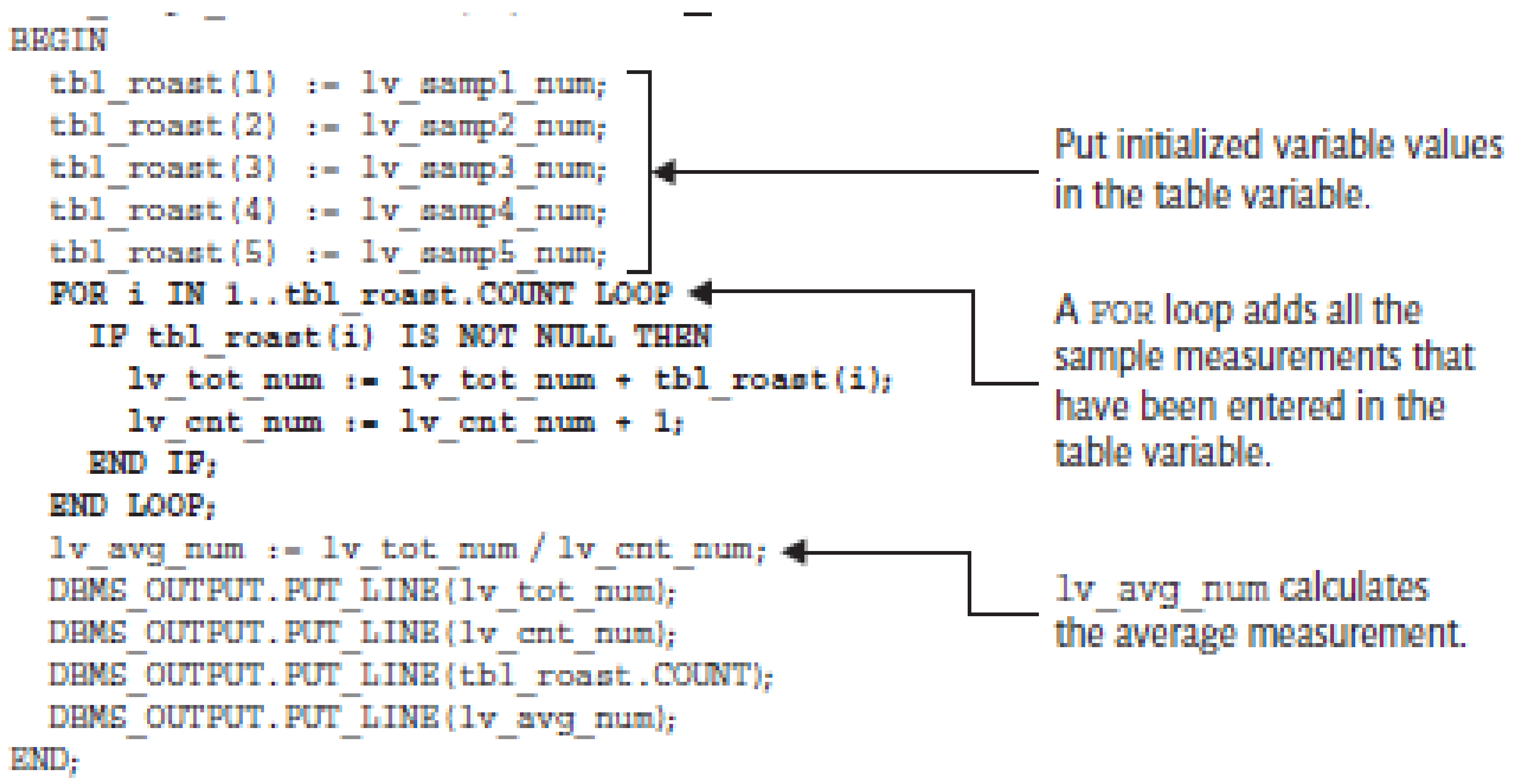
Associative Array Example


Associative array data type declaration
Associative array variable declaration
Declaring initialized variables

```
DECLARE
   TYPE type_roast IS TABLE OF NUMBER
   INDEX BY BINARY_INTEGER;
   tbl_roast type_roast;
   lv_tot_num NUMBER := 0;
   lv_cnt_num NUMBER := 0;
   lv_avg_num NUMBER;
   lv_samp1_num NUMBER(5,2) := 6.22;
   lv_samp2_num NUMBER(5,2) := 6.13;
   lv_samp3_num NUMBER(5,2) := 6.27;
   lv_samp4_num NUMBER(5,2) := 6.16;
   lv_samp5_num NUMBER(5,2);
```

Example (continued)

P
L
/
S
Q
L



```
BEGIN
  tbl_roast(1) := lv_samp1_num;
  tbl_roast(2) := lv_samp2_num;
  tbl_roast(3) := lv_samp3_num;
  tbl_roast(4) := lv_samp4_num;
  tbl_roast(5) := lv_samp5_num;
  FOR i IN 1..tbl_roast.COUNT LOOP
    IF tbl_roast(i) IS NOT NULL THEN
      lv_tot_num := lv_tot_num + tbl_roast(i);
      lv_cnt_num := lv_cnt_num + 1;
    END IF;
  END LOOP;
  lv_avg_num := lv_tot_num / lv_cnt_num;
  DBMS_OUTPUT.PUT_LINE(lv_tot_num);
  DBMS_OUTPUT.PUT_LINE(lv_cnt_num);
  DBMS_OUTPUT.PUT_LINE(tbl_roast.COUNT);
  DBMS_OUTPUT.PUT_LINE(lv_avg_num);
END;
```

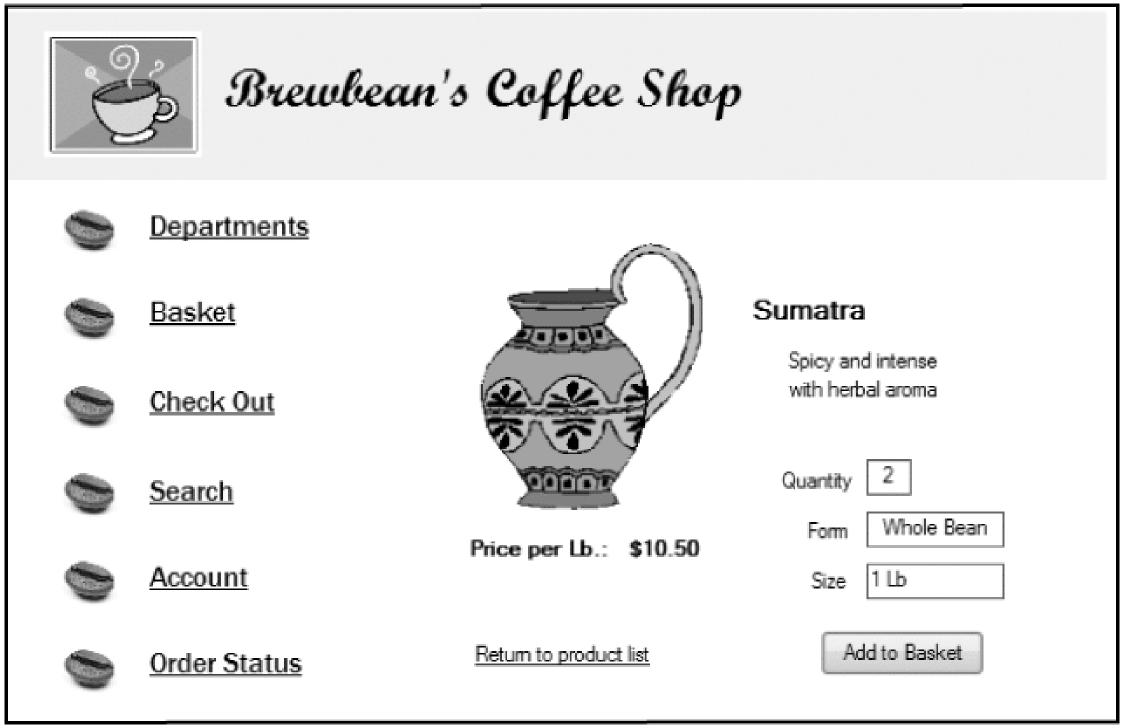
Put initialized variable values in the table variable.

A FOR loop adds all the sample measurements that have been entered in the table variable.

lv_avg_num calculates the average measurement.

Table of Records

- Contains one or more records
- Handle shopping basket data


Brewbean's Coffee Shop

Departments
Basket
Check Out
Search
Account
Order Status

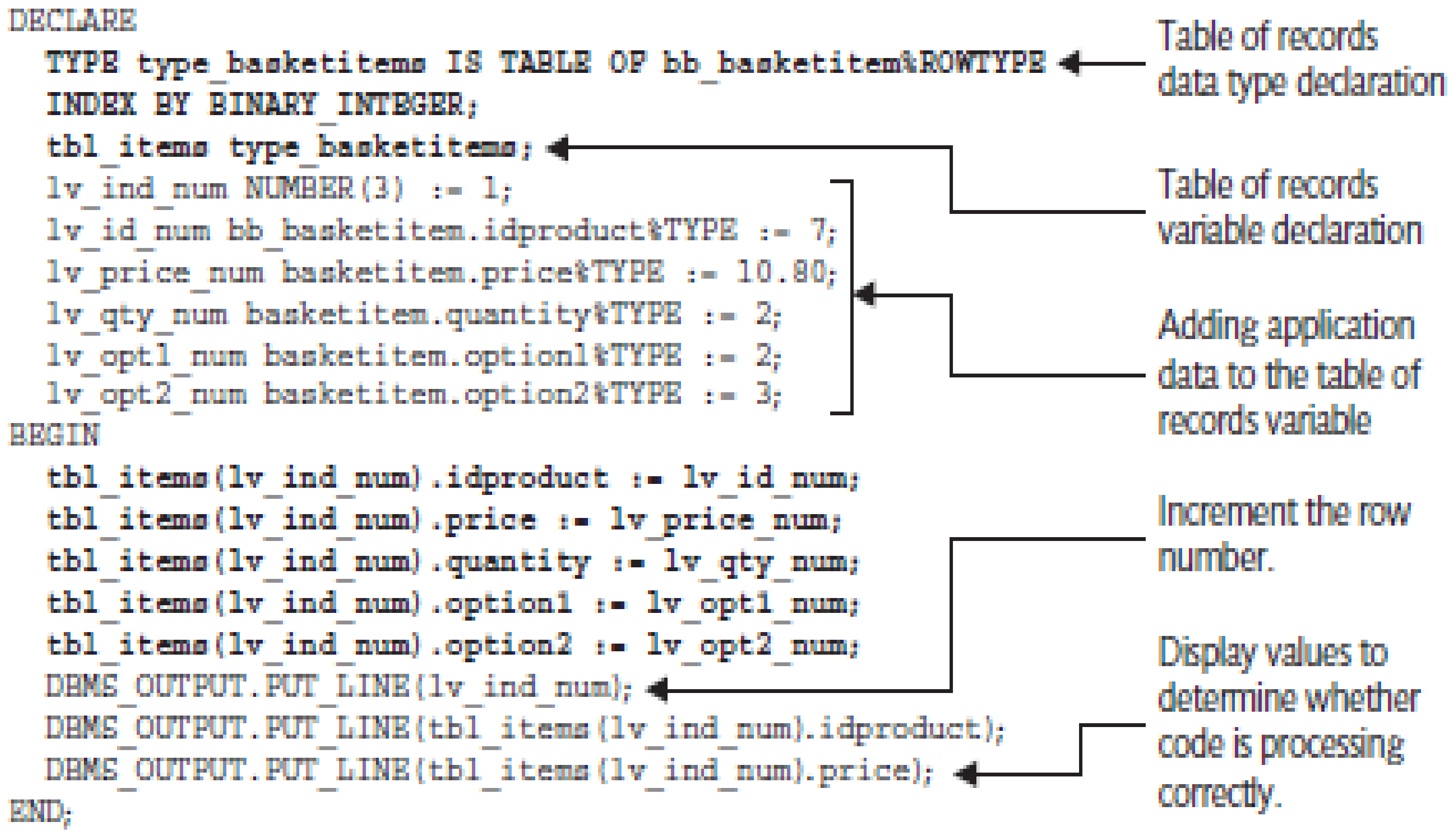
Sumatra

Spicy and intense with herbal aroma

Price per Lb.: $10.50

Quantity	2
Form	Whole Bean
Size	1 Lb

Table of Records



```
DECLARE
  TYPE type_basketitems IS TABLE OF bb_basketitem%ROWTYPE
  INDEX BY BINARY_INTEGER;
  tbl_items type_basketitems;
  lv_ind_num NUMBER(3) := 1;
  lv_id_num bb_basketitem.idproduct%TYPE := 7;
  lv_price_num basketitem.price%TYPE := 10.80;
  lv_qty_num basketitem.quantity%TYPE := 2;
  lv_opt1_num basketitem.option1%TYPE := 2;
  lv_opt2_num basketitem.option2%TYPE := 3;
BEGIN
  tbl_items(lv_ind_num).idproduct := lv_id_num;
  tbl_items(lv_ind_num).price := lv_price_num;
  tbl_items(lv_ind_num).quantity := lv_qty_num;
  tbl_items(lv_ind_num).option1 := lv_opt1_num;
  tbl_items(lv_ind_num).option2 := lv_opt2_num;
  DBMS_OUTPUT.PUT_LINE(lv_ind_num);
  DBMS_OUTPUT.PUT_LINE(tbl_items(lv_ind_num).idproduct);
  DBMS_OUTPUT.PUT_LINE(tbl_items(lv_ind_num).price);
END;
```

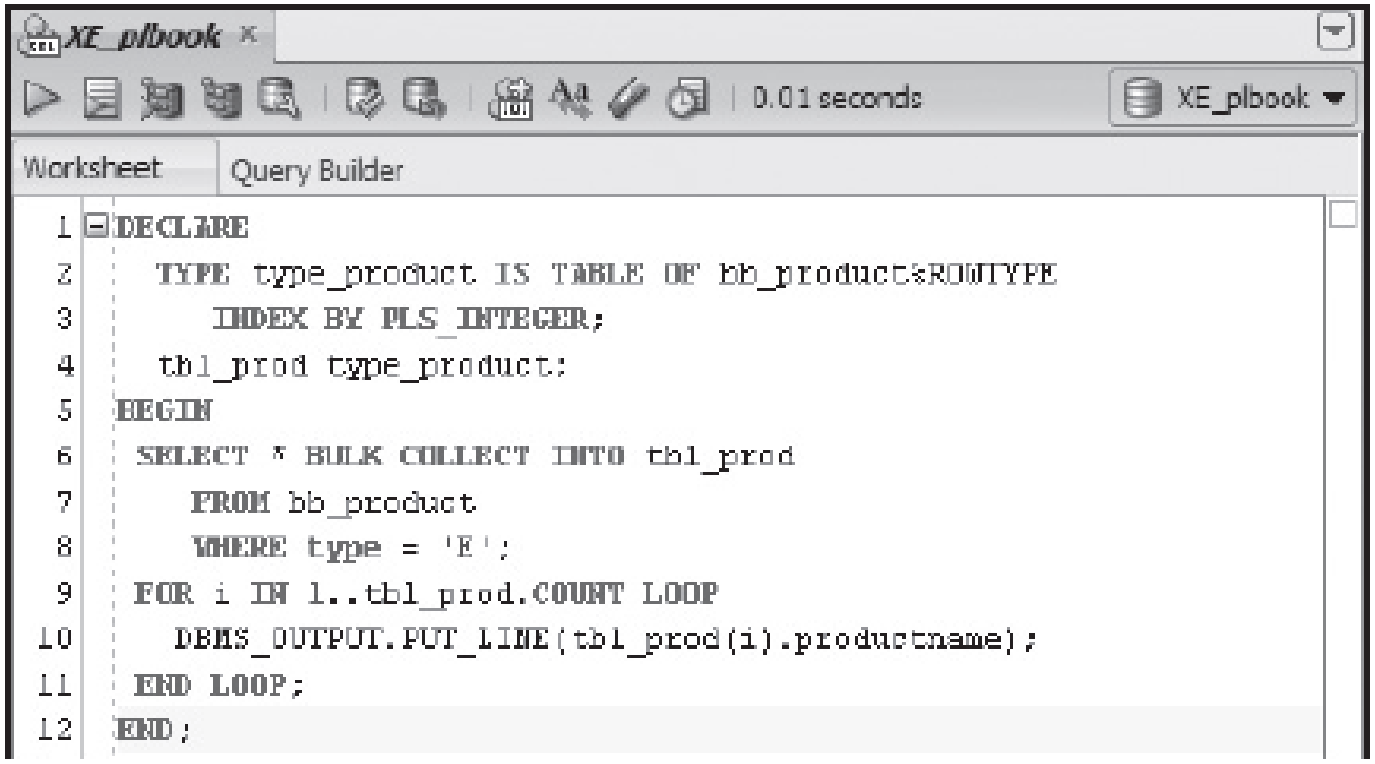
Table of records data type declaration
Table of records variable declaration
Adding application data to the table of records variable
Increment the row number.
Display values to determine whether code is processing correctly.

Bulk Processing

- Improve performance & add capabilities
- Reduces context switching
- Groups SQL actions for processing
- BULK COLLECT and FORALL statements
- More examples in Chapter 4

Bulk Processing

- Enables loading multi-row query directly to table of record variable



```
DECLARE
  TYPE type_product IS TABLE OF bb_product%ROWTYPE
     INDEX BY PLS_INTEGER;
  tbl_prod type_product;
BEGIN
 SELECT * BULK COLLECT INTO tbl_prod
    FROM bb_product
    WHERE type = 'E';
 FOR i IN 1..tbl_prod.COUNT LOOP
   DBMS_OUTPUT.PUT_LINE(tbl_prod(i).productname);
 END LOOP;
END;
```

GOTO Statement

- Jumping control that instructs the program to move to another area of code to continue processing
- Most developers discourage the use of GOTO as it complicates the flow of execution

Summary

P
L
/
S
Q
L

- SQL queries and DML statements can be embedded into a block
- An INTO clause must be added to a SELECT
- The %TYPE attribute is used to use a column data type
- Composite data types can hold multiple values in a single variable
- A record can hold a row of data
- A table of records can hold multiple rows of data

P
L
/
S
Q
L

Summary (continued)

- The %ROWTYPE attribute can be used to declare a data type based on a table's structure
- An associative array is a collection of same type data
- Bulk processing groups SQL statements for processing to improve performance
- The GOTO statement enables execution to jump to specific portions of code

P
L
/
S
Q
L

Oracle:
PL/SQL Programming

Chapter 4

Cursors and Exception Handling

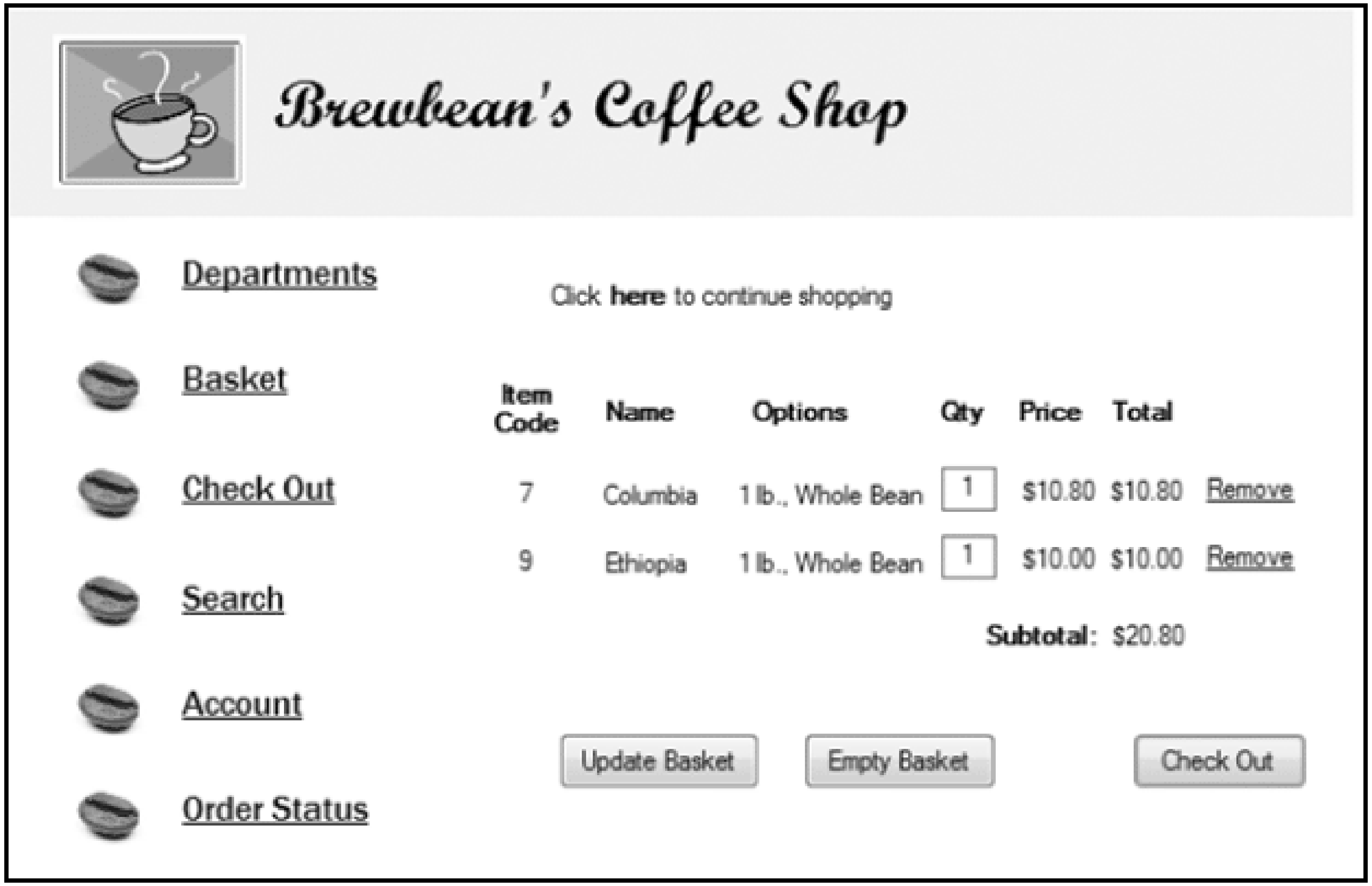
Chapter Objectives

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

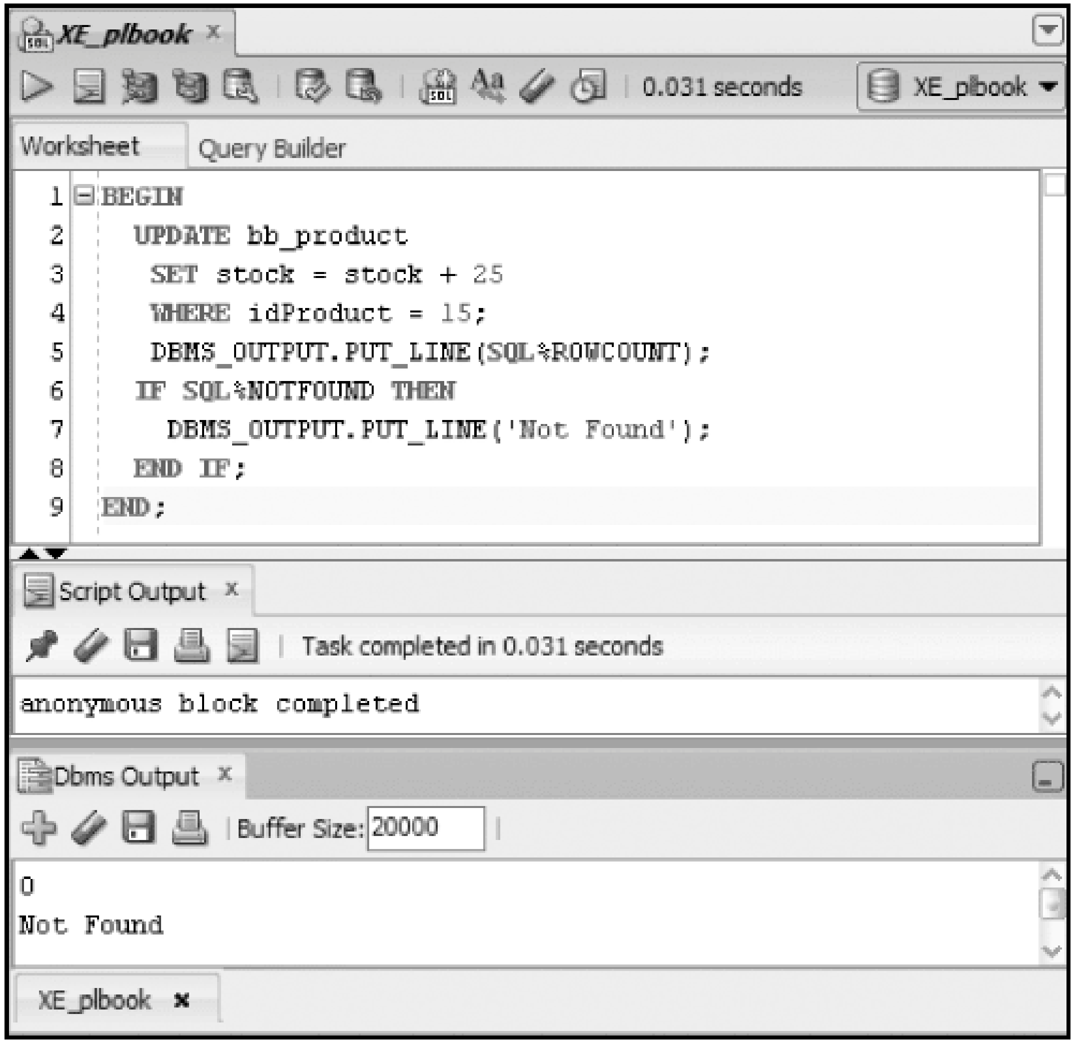
- Work area in which SQL statement is processed
- Implicit cursor – declared automatically for DML and SELECT statements
- Explicit cursor – declared and managed programmatically to handle a set of rows returned by a SELECT statement
- Cursor variable – reference or pointer to a work area or cursor

P
L
/
S
Q
L

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



```
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
```

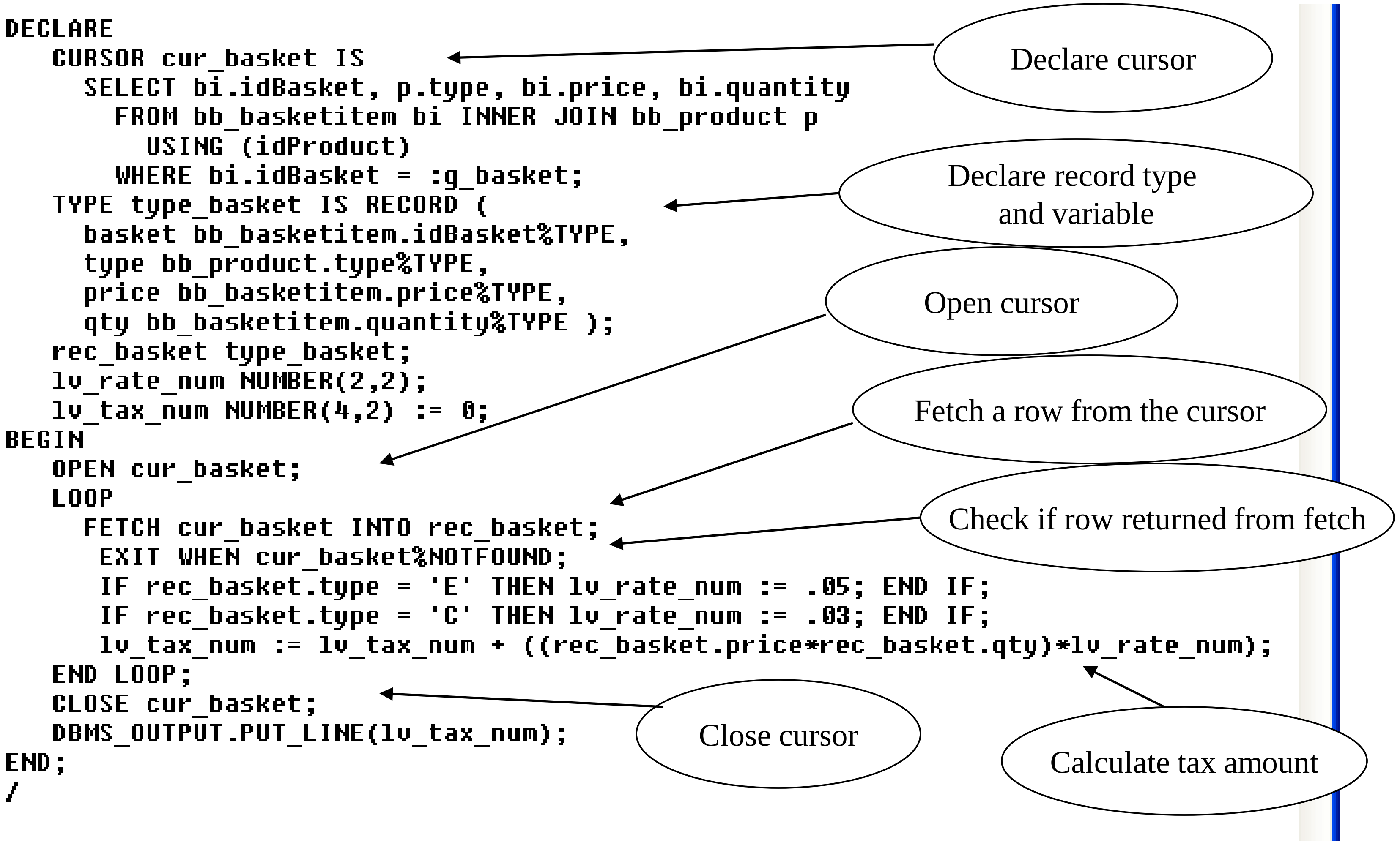
```
0
Not Found
```

Explicit Cursor

P
L
/
S
Q
L

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

```
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;
/
```

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

Cursor FOR Loop Example

P
L
/
S
Q
L

```
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

P
L
/
S
Q
L

- 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

```
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;
```

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

```
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)

```
  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

P L / S Q L

- 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

PL/SQL

Bulk-processing (Query)

```
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)

P
L
/
S
Q
L

```
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

P
L
/
S
Q
L

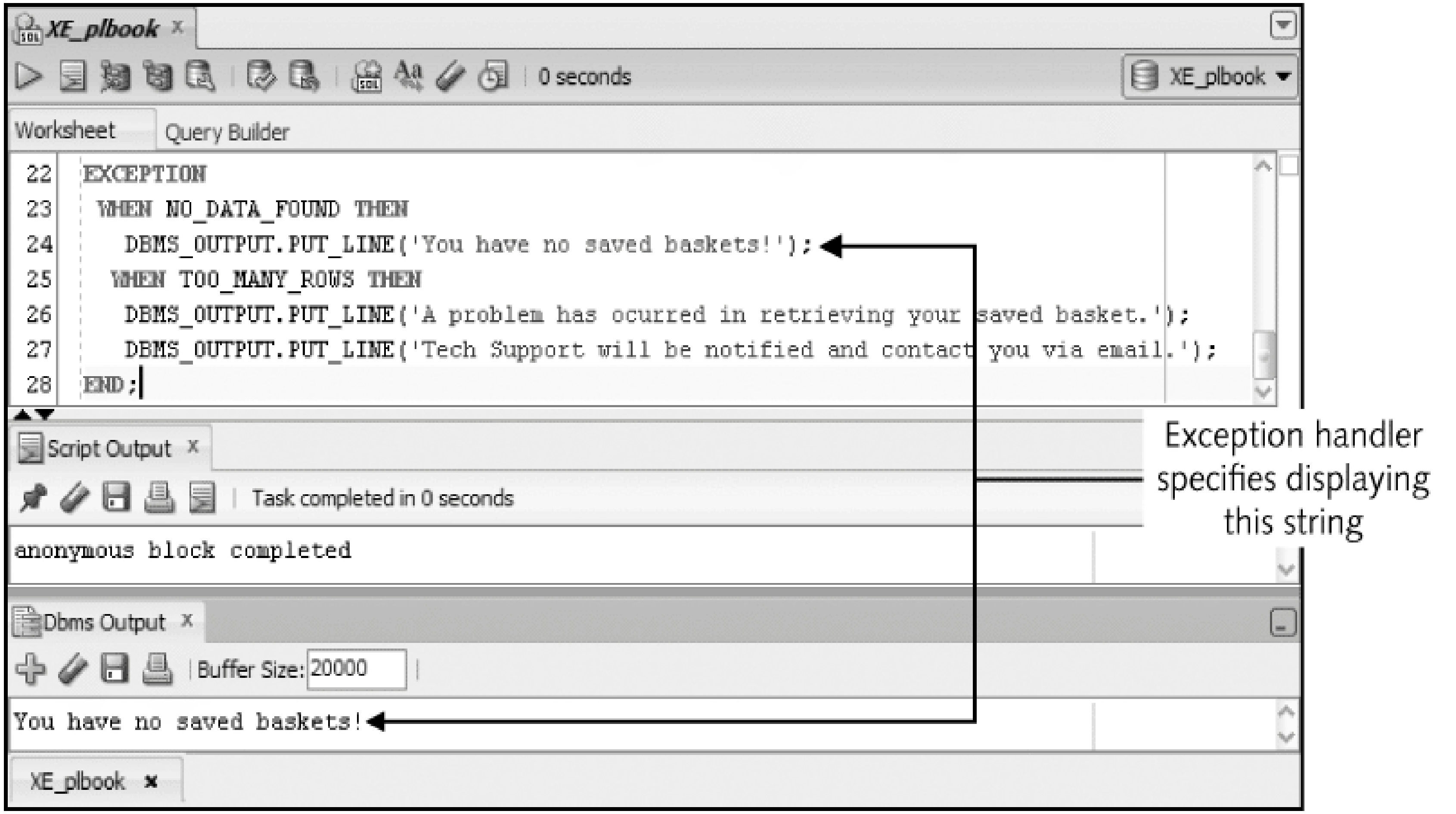
- Used to capture error conditions and handle the processing to allow the application to continue
- Placed in the EXCEPTION section of a PL/SQL block
- Two types of errors
    1. Oracle errors (Predefined and Non-Predefined)
    2. User-defined errors
- RAISE_APPLICATION_ERROR

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



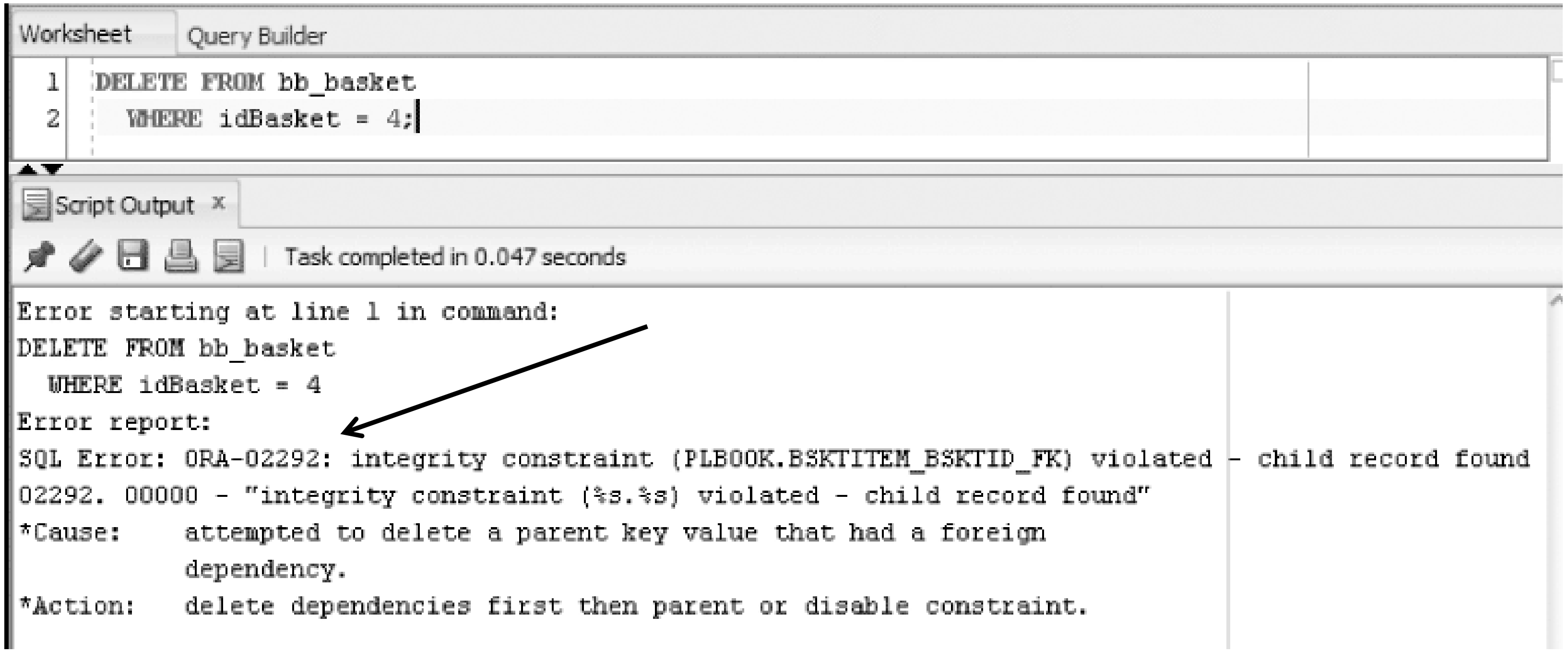
```
EXCEPTION
 WHEN NO_DATA_FOUND THEN
   DBMS_OUTPUT.PUT_LINE('You have no saved baskets!');
  WHEN TOO_MANY_ROWS THEN
   DBMS_OUTPUT.PUT_LINE('A problem has ocurred in retrieving your saved basket.');
   DBMS_OUTPUT.PUT_LINE('Tech Support will be notified and contact you via email.');
END;
```

```
anonymous block completed
```

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

Undefined Error

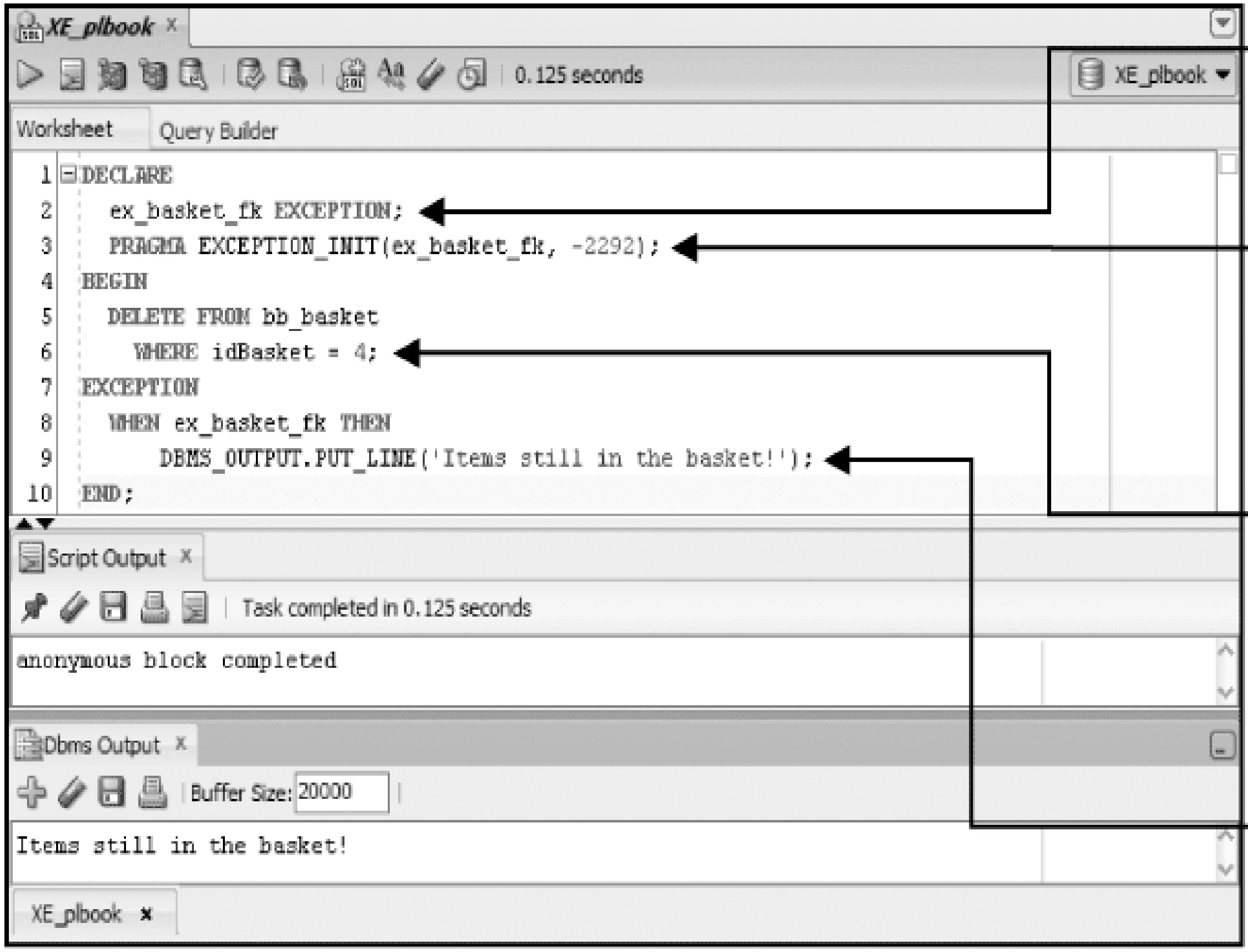
- Identify possible errors for statements in a block



```
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

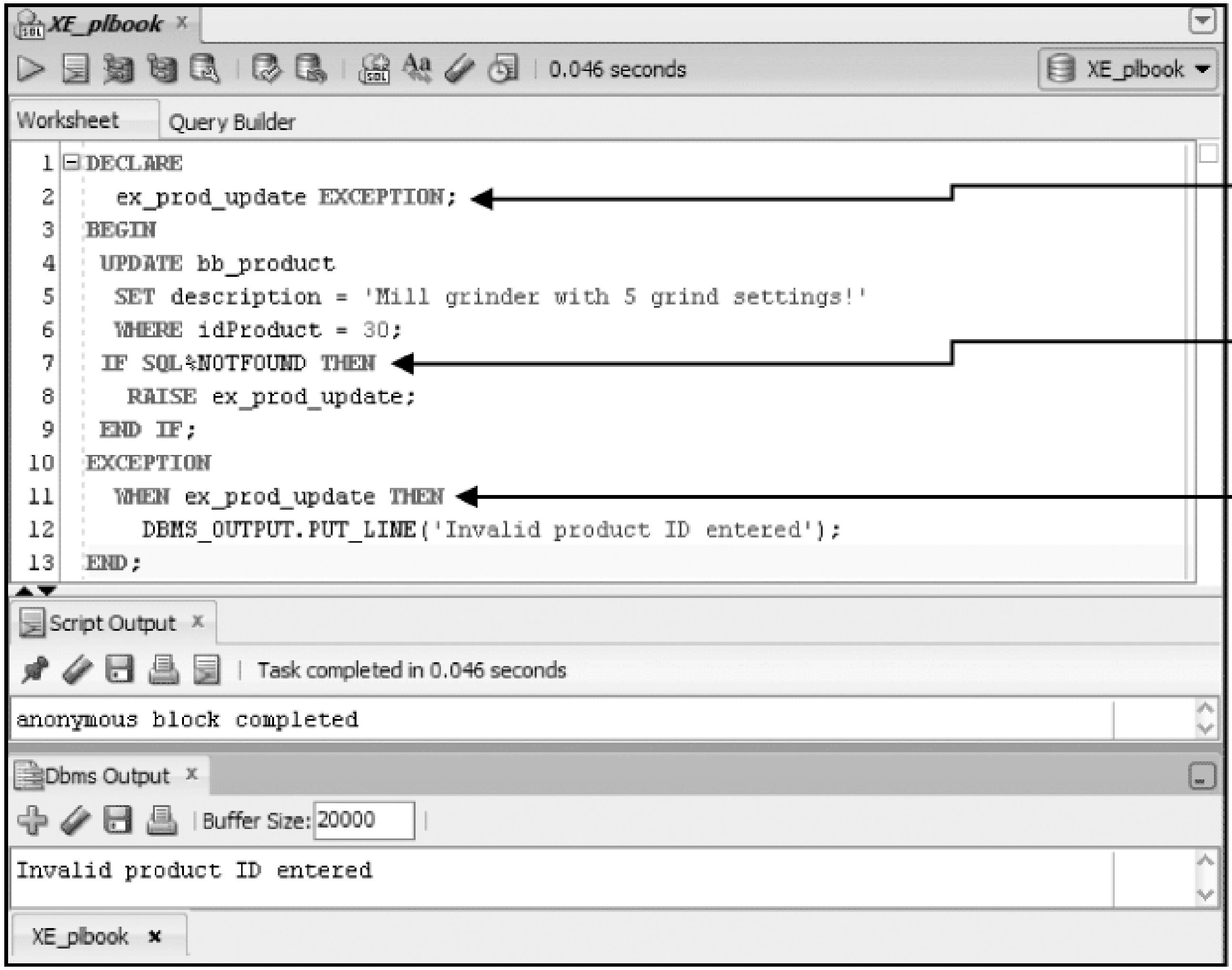
Exception handler runs if the DELETE statement raises foreign key error -2292

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

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
```

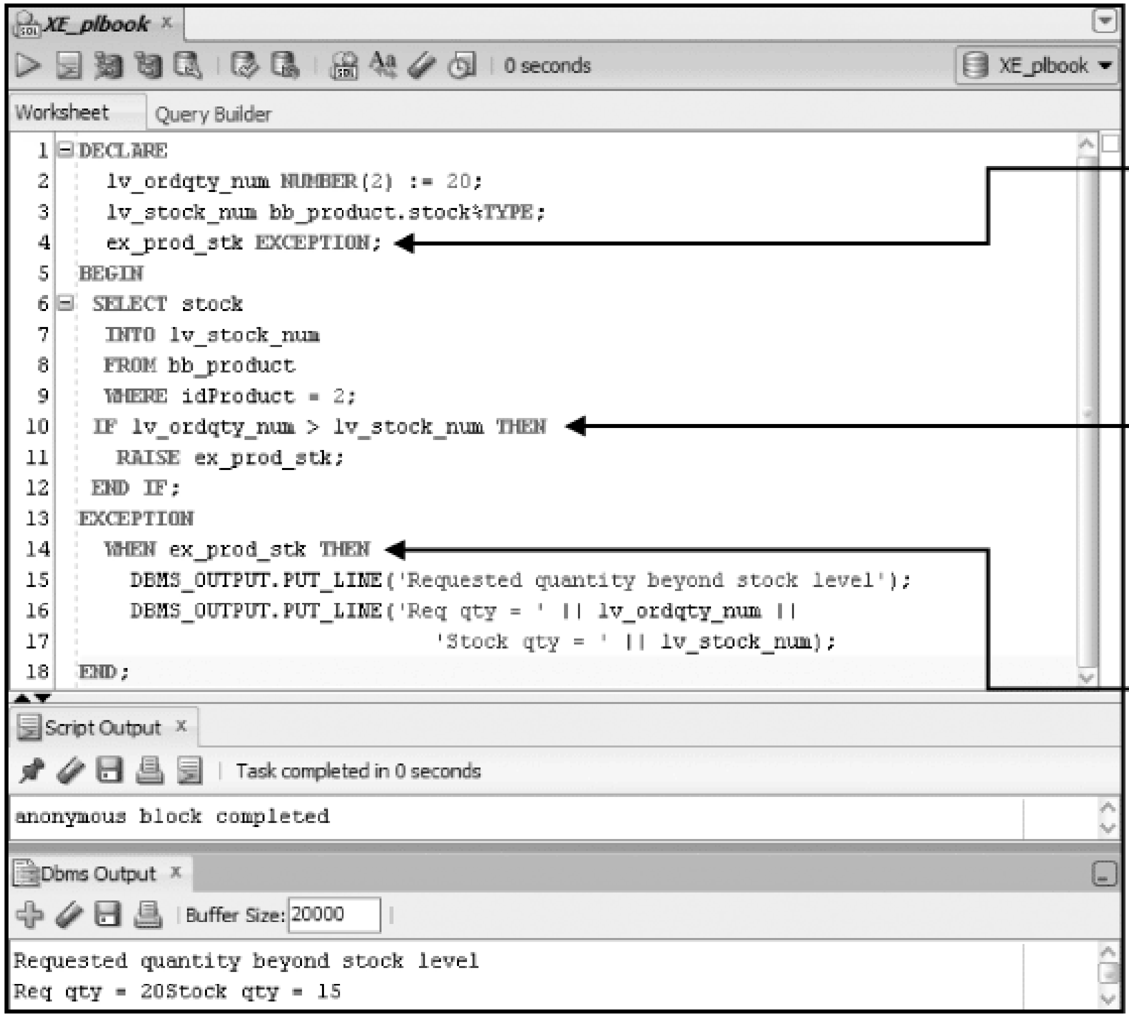
```
Invalid product ID entered
```

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

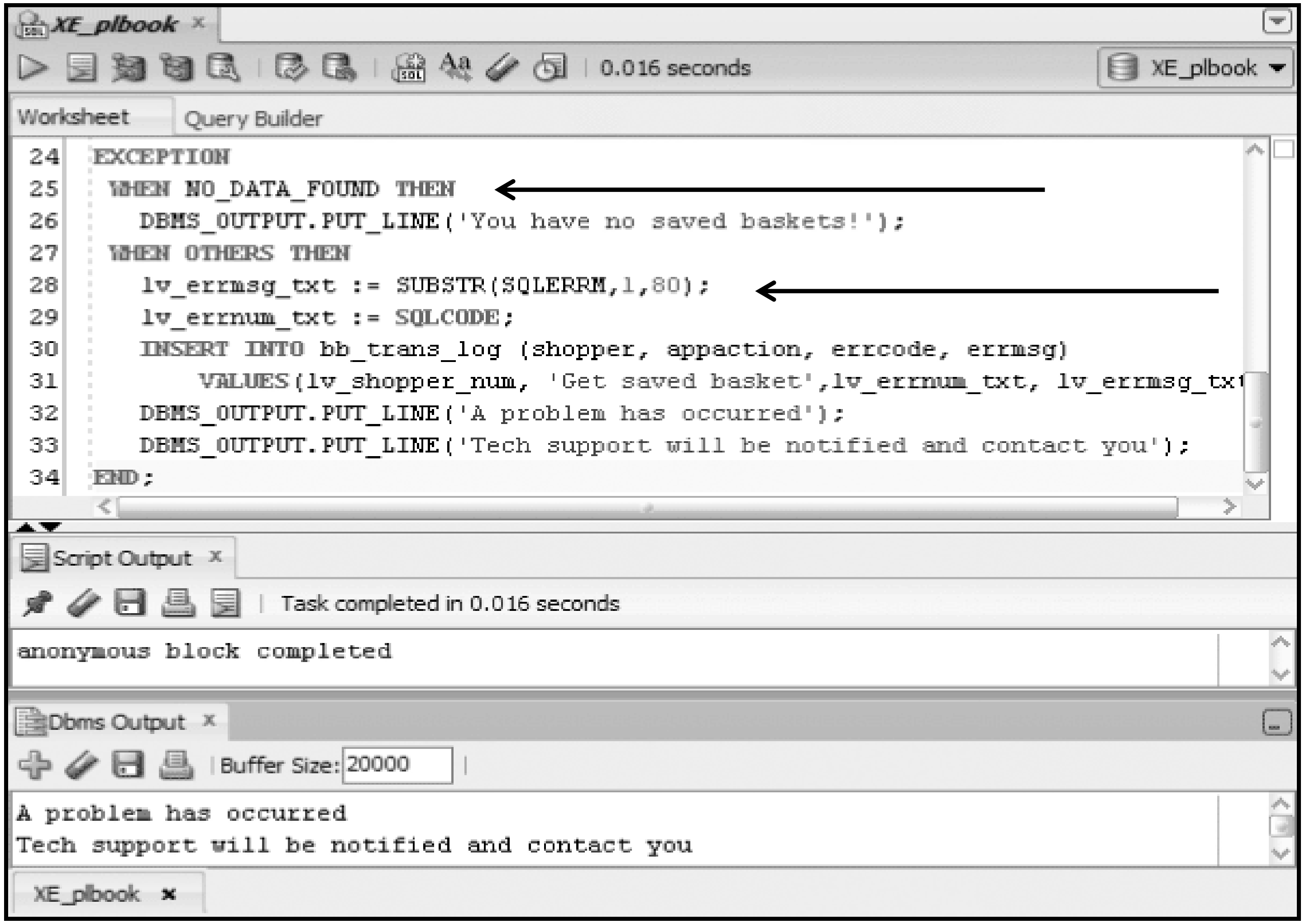
Additional Exception Concepts

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

P
L
/
S
Q
L



XE_plbook

0.016 seconds

Worksheet Query Builder

```
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;
```

Script Output

Task completed in 0.016 seconds

```
anonymous block completed
```

Dbms Output

Buffer Size: 20000

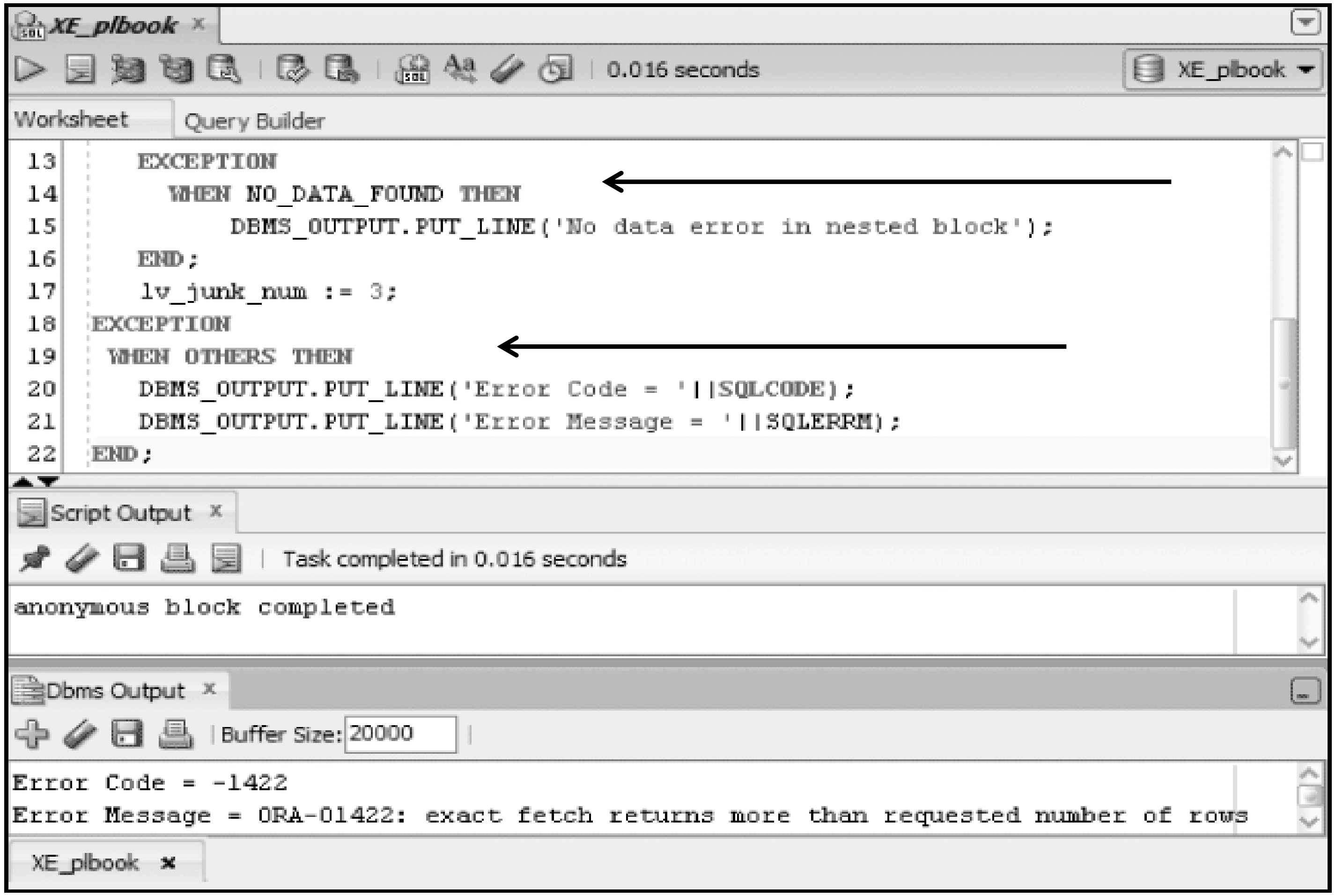
```
A problem has occurred
Tech support will be notified and contact you
```

XE_plbook

Exception Propagation

- 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



```
   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

P
L
/
S
Q
L

- 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

P
L
/
S
Q
L

Oracle:
PL/SQL Programming

Chapter 5

Procedures

Chapter Objectives

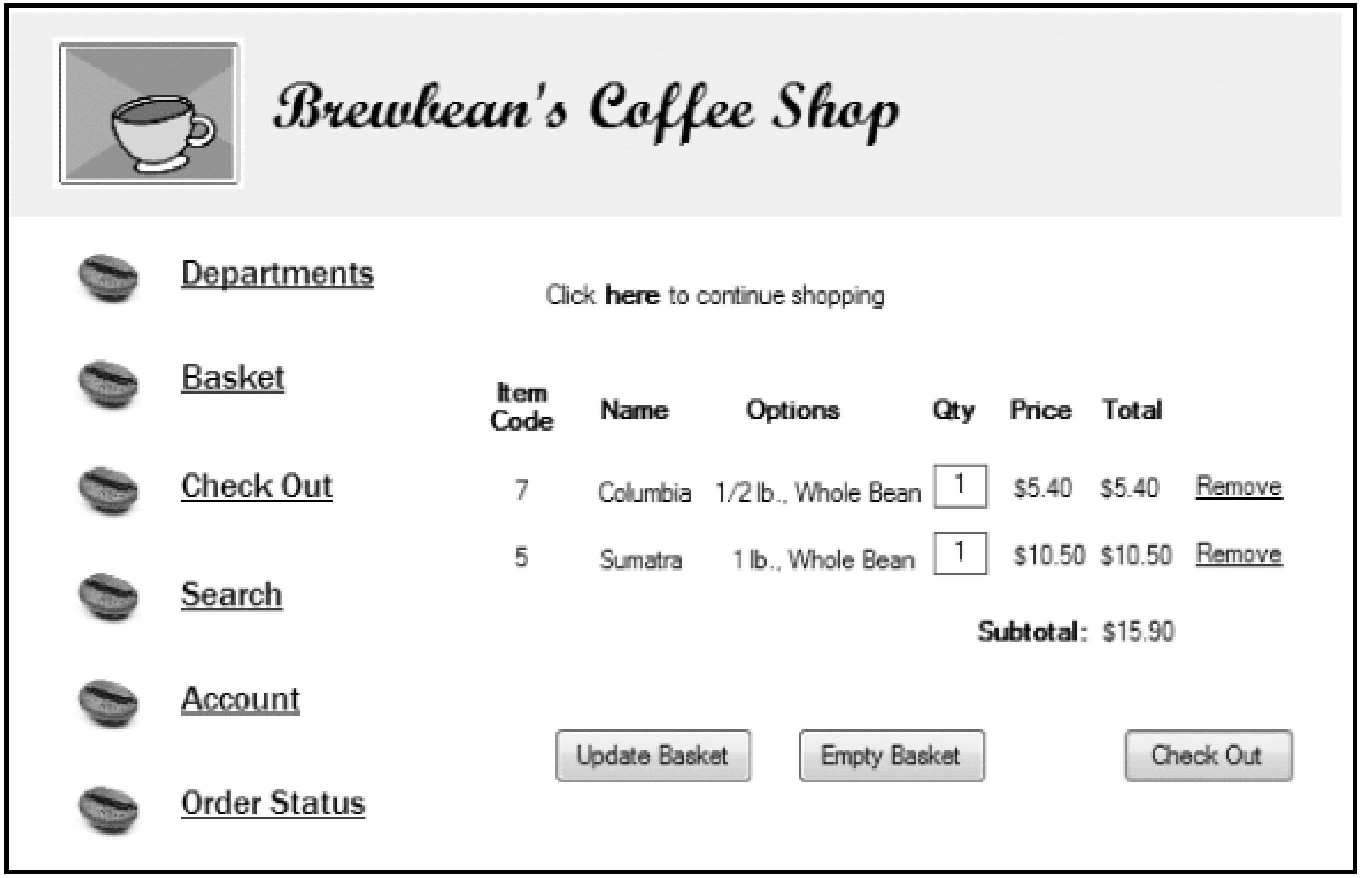
- After completing this lesson, you should be able to understand:
  - Named program units
  - Creating a procedure
  - Calling a procedure from another procedure
  - Using the DESCRIBE command with procedures
  - Debugging procedures using DBMS_OUTPUT

Chapter Objectives (continued)

- After completing this lesson, you should be able to understand (continued):
  - Using subprograms
  - The scope of variables, exception handling and transaction control
  - Using RAISE_APPLICATION_ERROR for error handling
  - Removing procedures

Brewbean's Challenge

- Develop programming modules for specific tasks such as calculating taxes or updating inventory



Brewbean's Coffee Shop

Departments
Basket
Check Out
Search
Account
Order Status

Click **here** to continue shopping

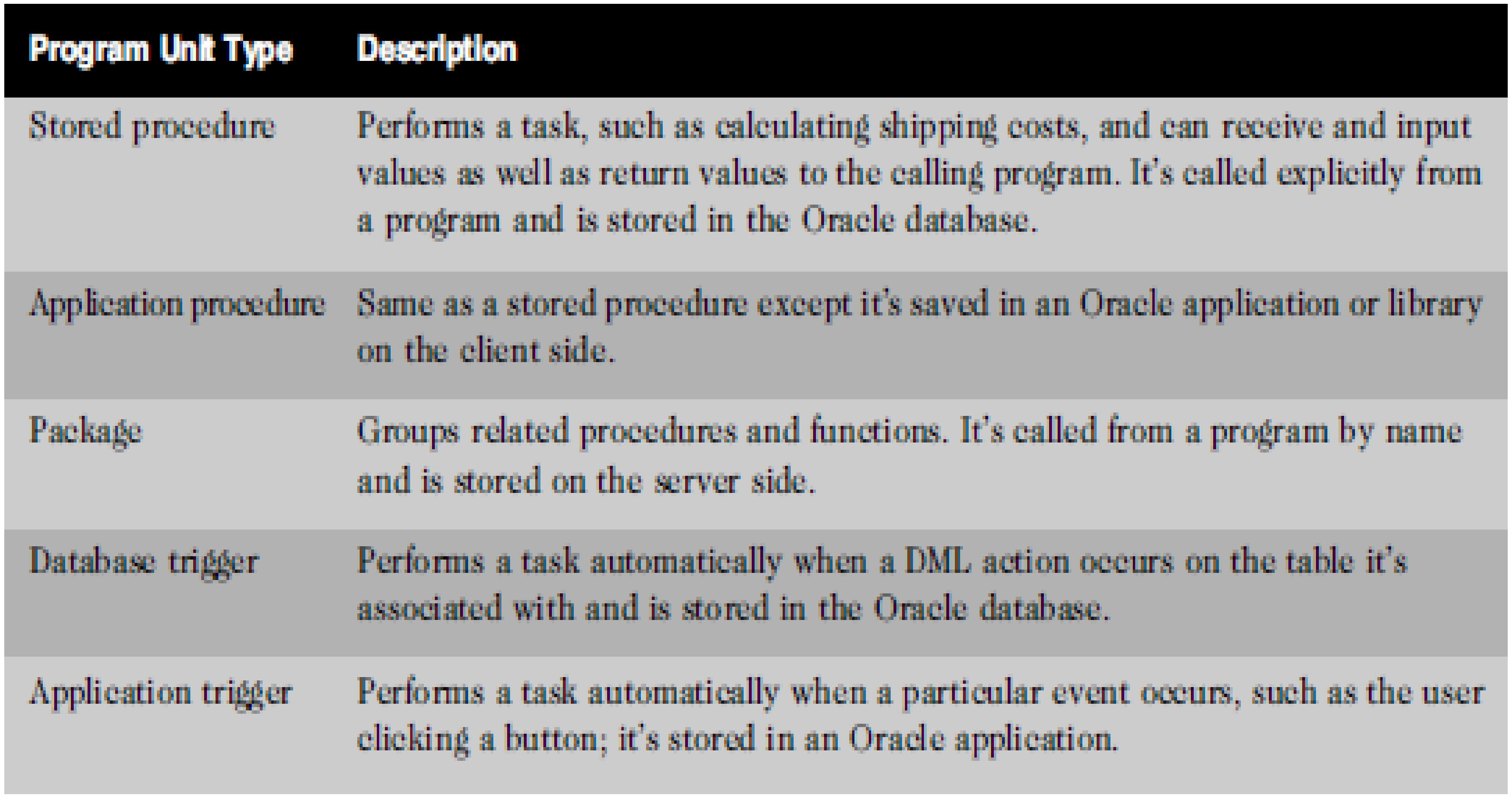
Item Code	Name	Options	Qty	Price	Total	
7	Columbia	1/2 lb., Whole Bean	1	$5.40	$5.40	Remove
5	Sumatra	1 lb., Whole Bean	1	$10.50	$10.50	Remove

Subtotal: $15.90

Named Program Units

- PL/SQL blocks executed thus far have been anonymous blocks
- Now we will assign a name to the block and save it in the database as a stored program unit
- This makes program units reusable

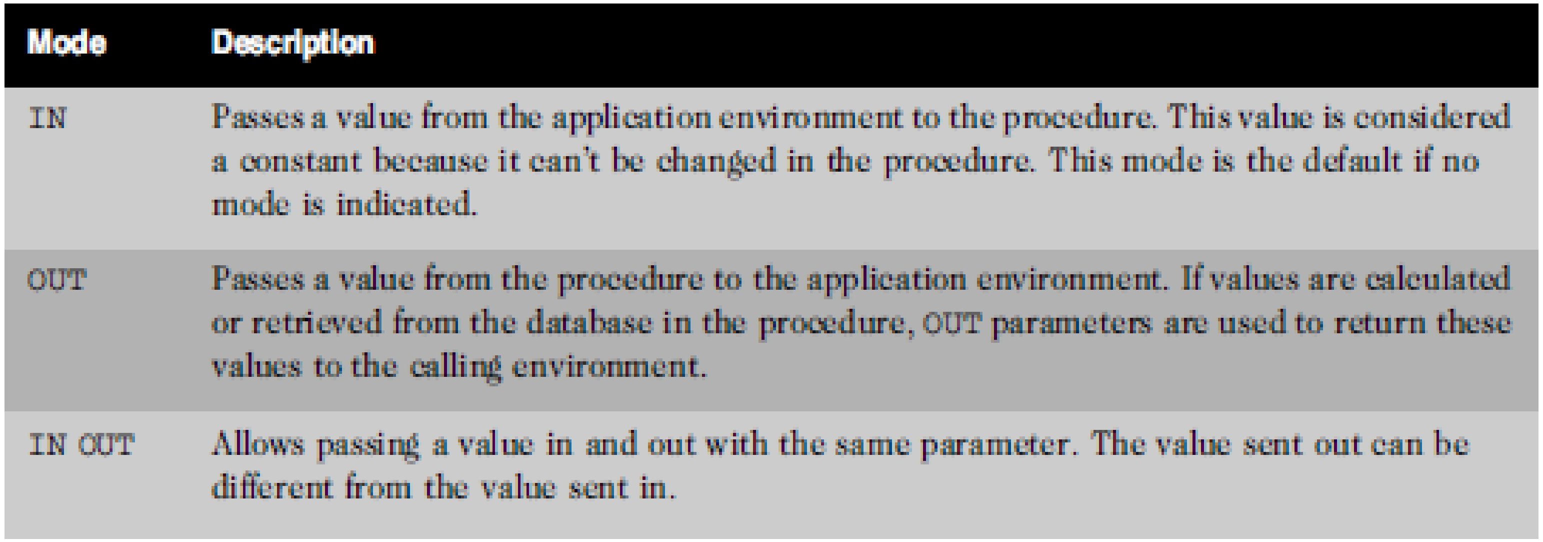
Types of Program Units



Program Unit Type	Description
Stored procedure	Performs a task, such as calculating shipping costs, and can receive and input values as well as return values to the calling program. It's called explicitly from a program and is stored in the Oracle database.
Application procedure	Same as a stored procedure except it's saved in an Oracle application or library on the client side.
Package	Groups related procedures and functions. It's called from a program by name and is stored on the server side.
Database trigger	Performs a task automatically when a DML action occurs on the table it's associated with and is stored in the Oracle database.
Application trigger	Performs a task automatically when a particular event occurs, such as the user clicking a button; it's stored in an Oracle application.

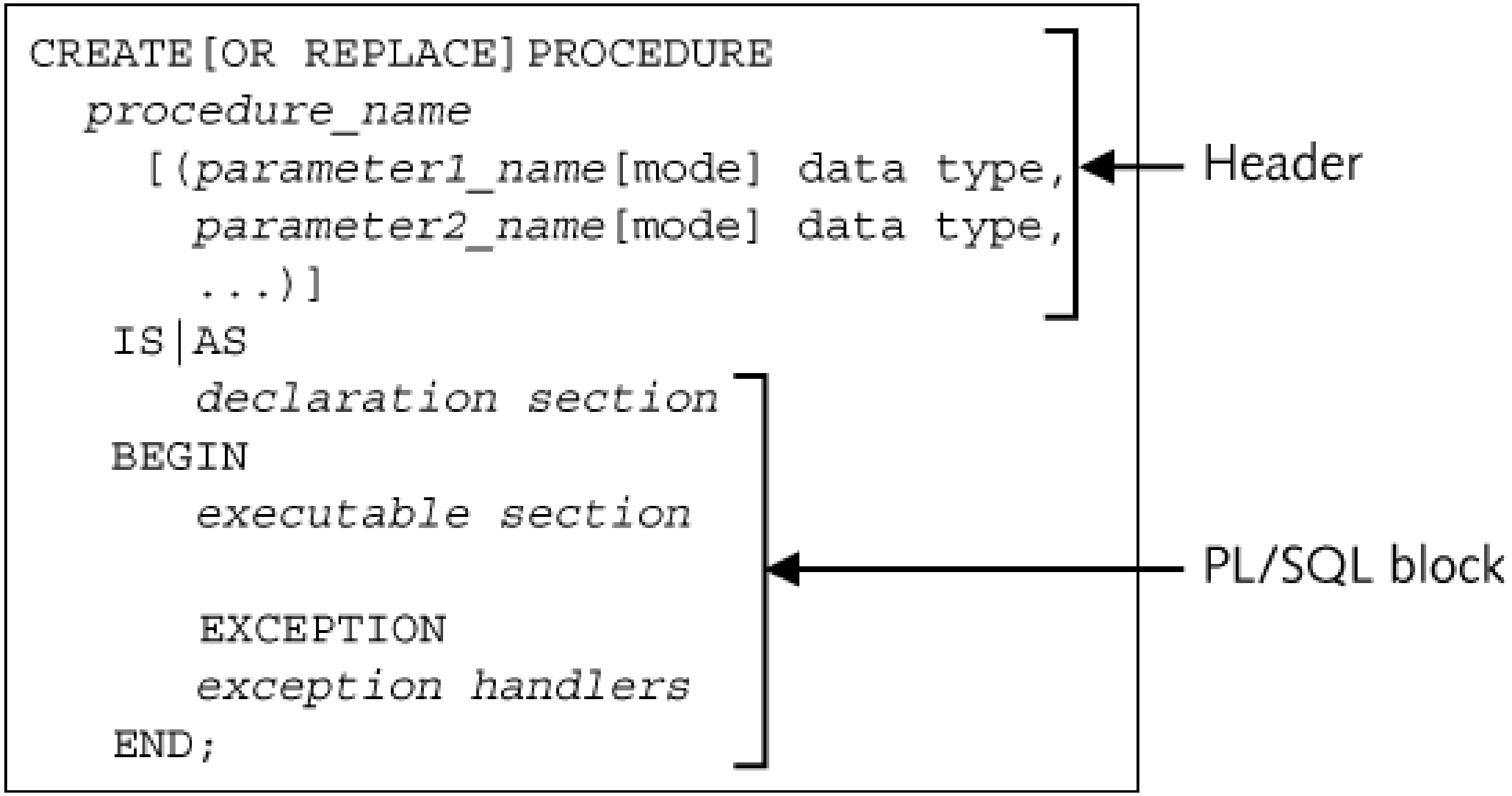
Parameters – Make Program Units Reusable

- Mechanisms used to send values in and out of program units



Mode	Description
IN	Passes a value from the application environment to the procedure. This value is considered a constant because it can't be changed in the procedure. This mode is the default if no mode is indicated.
OUT	Passes a value from the procedure to the application environment. If values are calculated or retrieved from the database in the procedure, OUT parameters are used to return these values to the calling environment.
IN OUT	Allows passing a value in and out with the same parameter. The value sent out can be different from the value sent in.

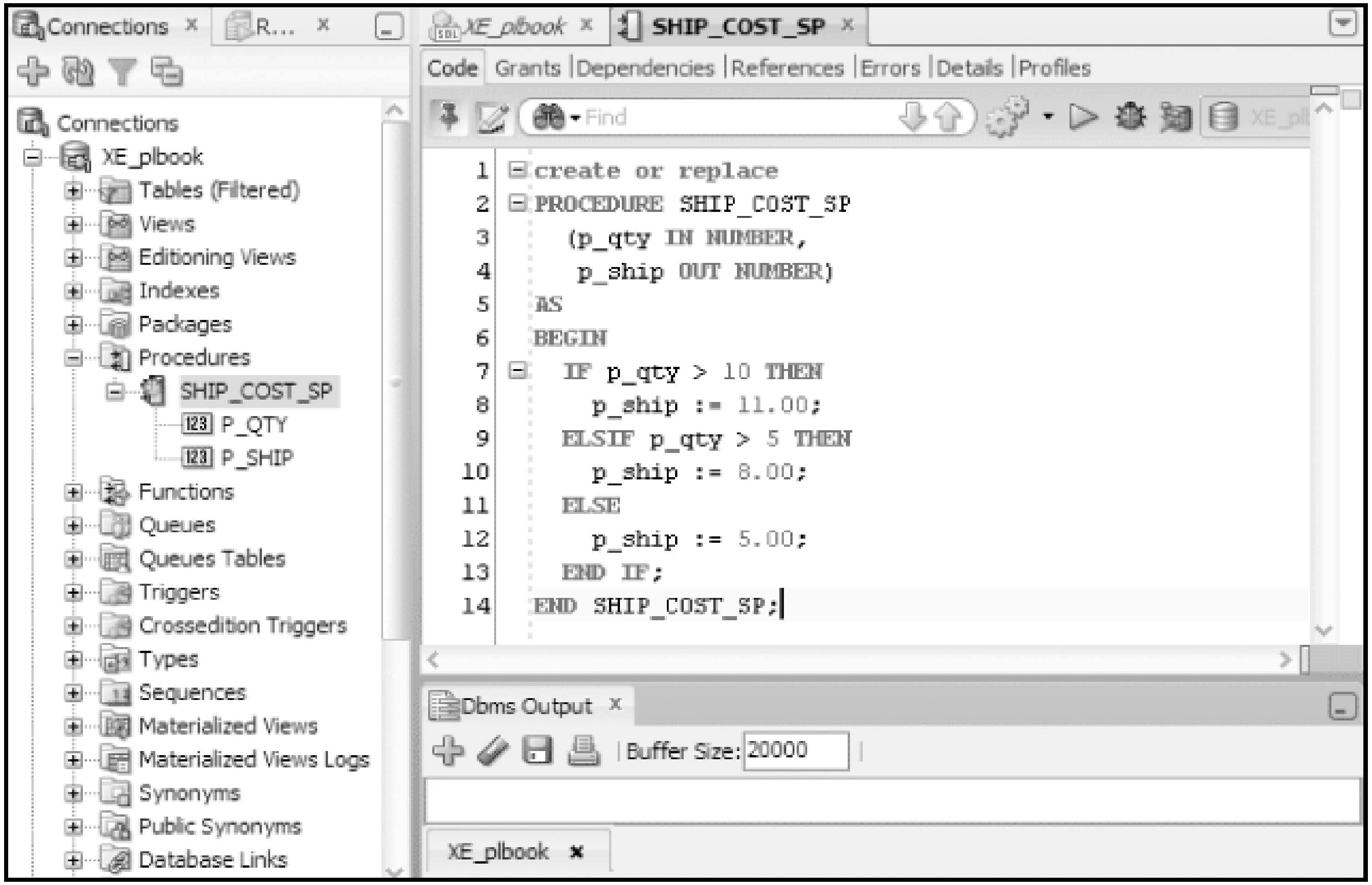
Create Procedure Statement Syntax



```
CREATE[OR REPLACE]PROCEDURE
  procedure_name
    [(parameter1_name[mode] data type,
      parameter2_name[mode] data type,
      ...)]
   IS|AS
      declaration section
   BEGIN
      executable section

      EXCEPTION
      exception handlers
   END;
```

Create Procedure Execution

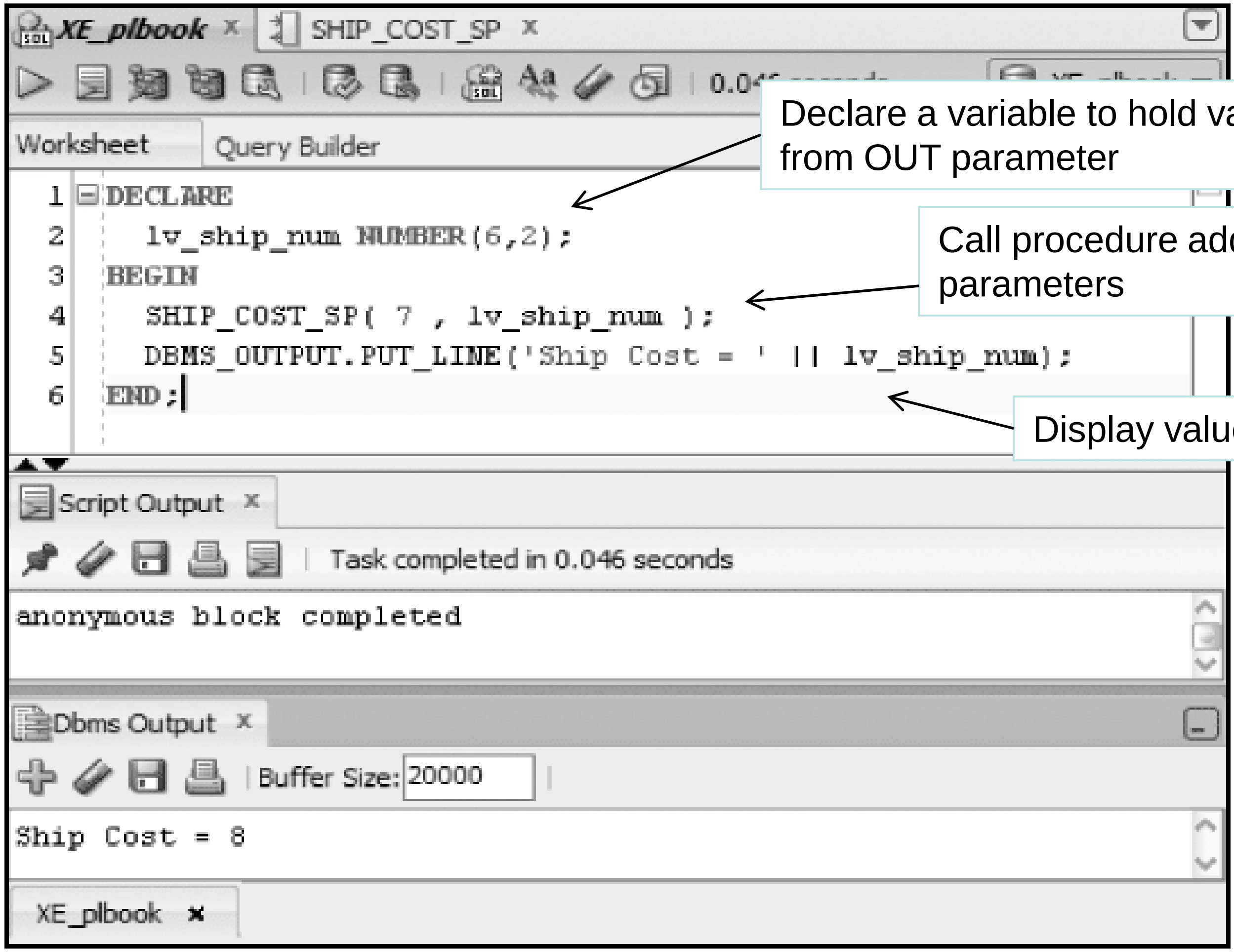
- Procedure to determine shipping cost



```
create or replace
PROCEDURE SHIP_COST_SP
  (p_qty IN NUMBER,
   p_ship OUT NUMBER)
AS
BEGIN
  IF p_qty > 10 THEN
    p_ship := 11.00;
  ELSIF p_qty > 5 THEN
    p_ship := 8.00;
  ELSE
    p_ship := 5.00;
  END IF;
END SHIP_COST_SP;
```

P
L
/
S
Q
L

Execute/Test the Procedure



```
DECLARE
  lv_ship_num NUMBER(6,2);
BEGIN
  SHIP_COST_SP( 7 , lv_ship_num );
  DBMS_OUTPUT.PUT_LINE('Ship Cost = ' || lv_ship_num);
END;
```

```
anonymous block completed
```

```
Ship Cost = 8
```

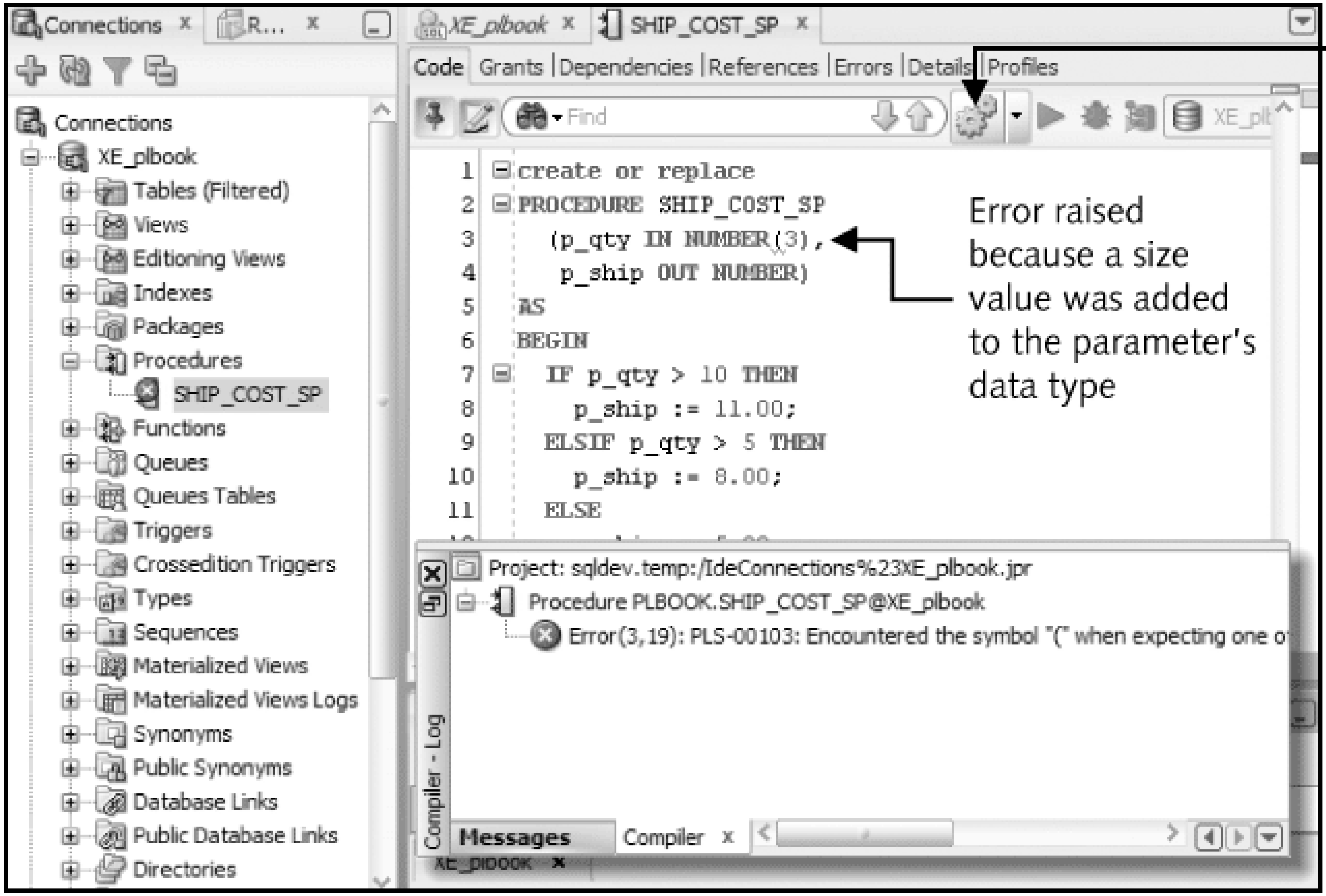
Declare a variable to hold value from OUT parameter

Call procedure addressing both parameters

Display value returned to verify

Note: Parameter arguments are passed positionally by default

Compilation errors



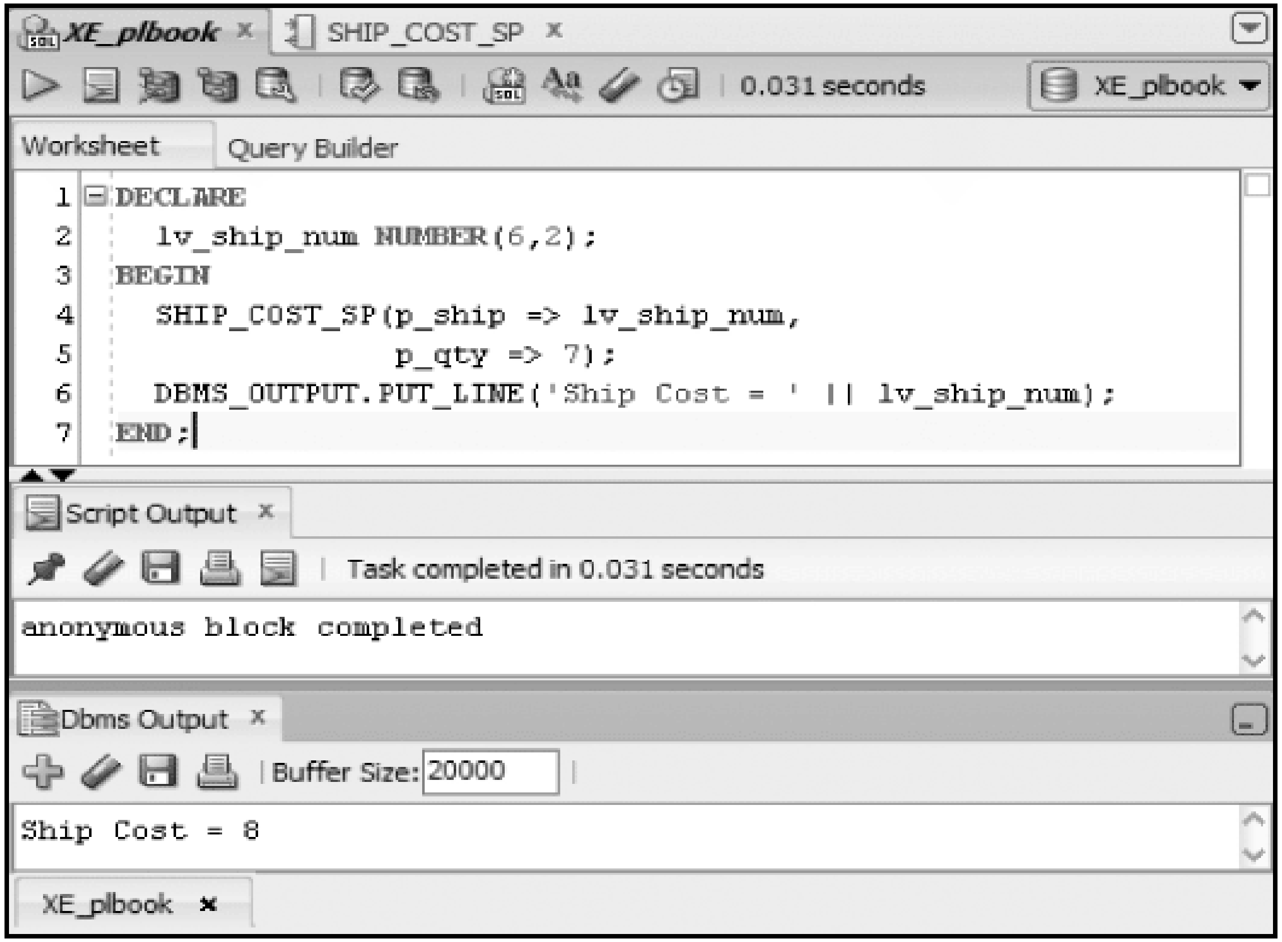
```
create or replace
PROCEDURE SHIP_COST_SP
  (p_qty IN NUMBER(3),
   p_ship OUT NUMBER)
AS
BEGIN
  IF p_qty > 10 THEN
    p_ship := 11.00;
  ELSIF p_qty > 5 THEN
    p_ship := 8.00;
  ELSE
```

```
Project: sqldev.temp:/IdeConnections%23XE_plbook.jpr
  Procedure PLBOOK.SHIP_COST_SP@XE_plbook
    Error(3,19): PLS-00103: Encountered the symbol "(" when expecting one o
```

Named Association Method

P
L
/
S
Q
L

- Provide parameter values by position (default) or name



```
DECLARE
  lv_ship_num NUMBER(6,2);
BEGIN
  SHIP_COST_SP(p_ship => lv_ship_num,
               p_qty => 7);
  DBMS_OUTPUT.PUT_LINE('Ship Cost = ' || lv_ship_num);
END;
```

```
anonymous block completed
```

```
Ship Cost = 8
```

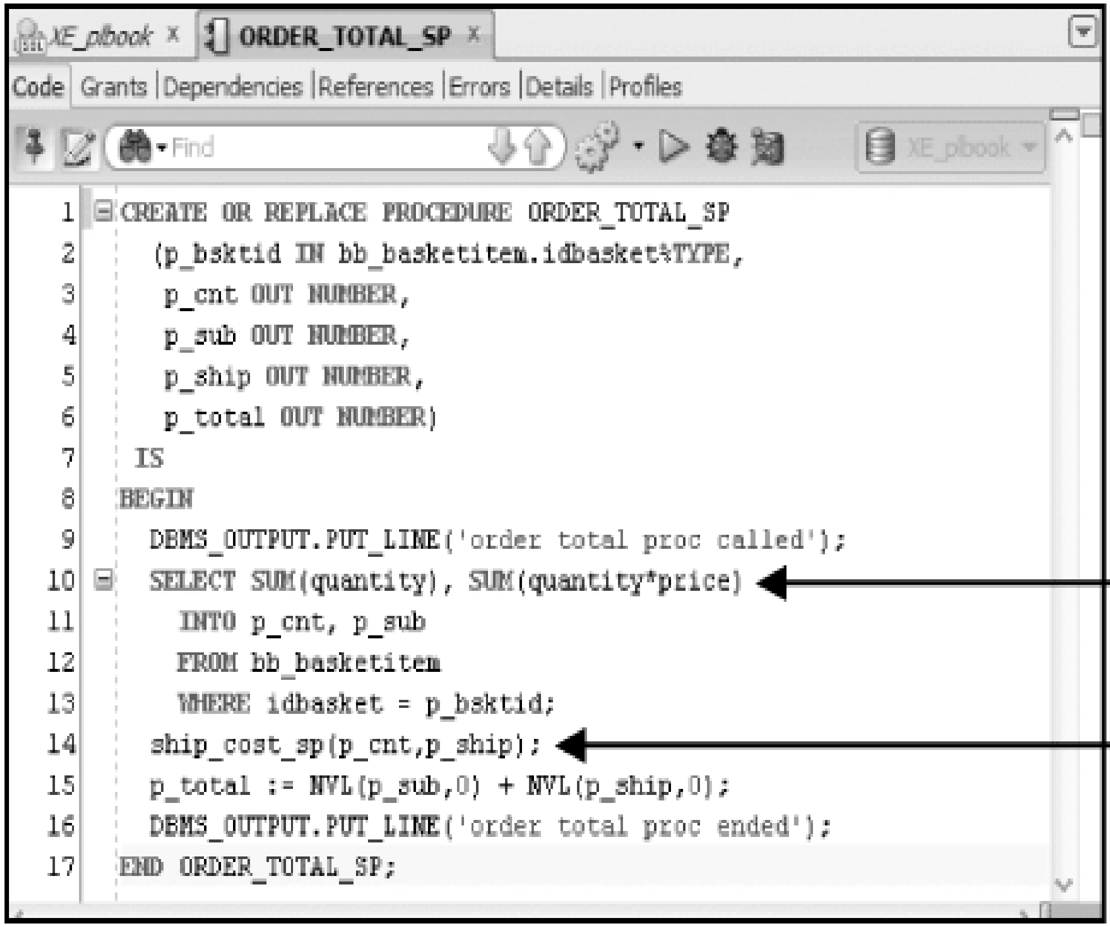
IN OUT mode

- Send value in and out via the same parameter

```
CREATE OR REPLACE PROCEDURE phone_fmt_sp
   (p_phone IN OUT VARCHAR2)
  IS
BEGIN
  p_phone := '(' || SUBSTR(p_phone,1,3) || ')' ||
                    SUBSTR(p_phone,4,3) || '-' ||
                    SUBSTR(p_phone,7,4);
END;
```

P
L
/
S
Q
L

Calling a Procedure from another procedure


XE_plbook
ORDER_TOTAL_SP
Code | Grants | Dependencies | References | Errors | Details | Profiles
Find
XE_plbook

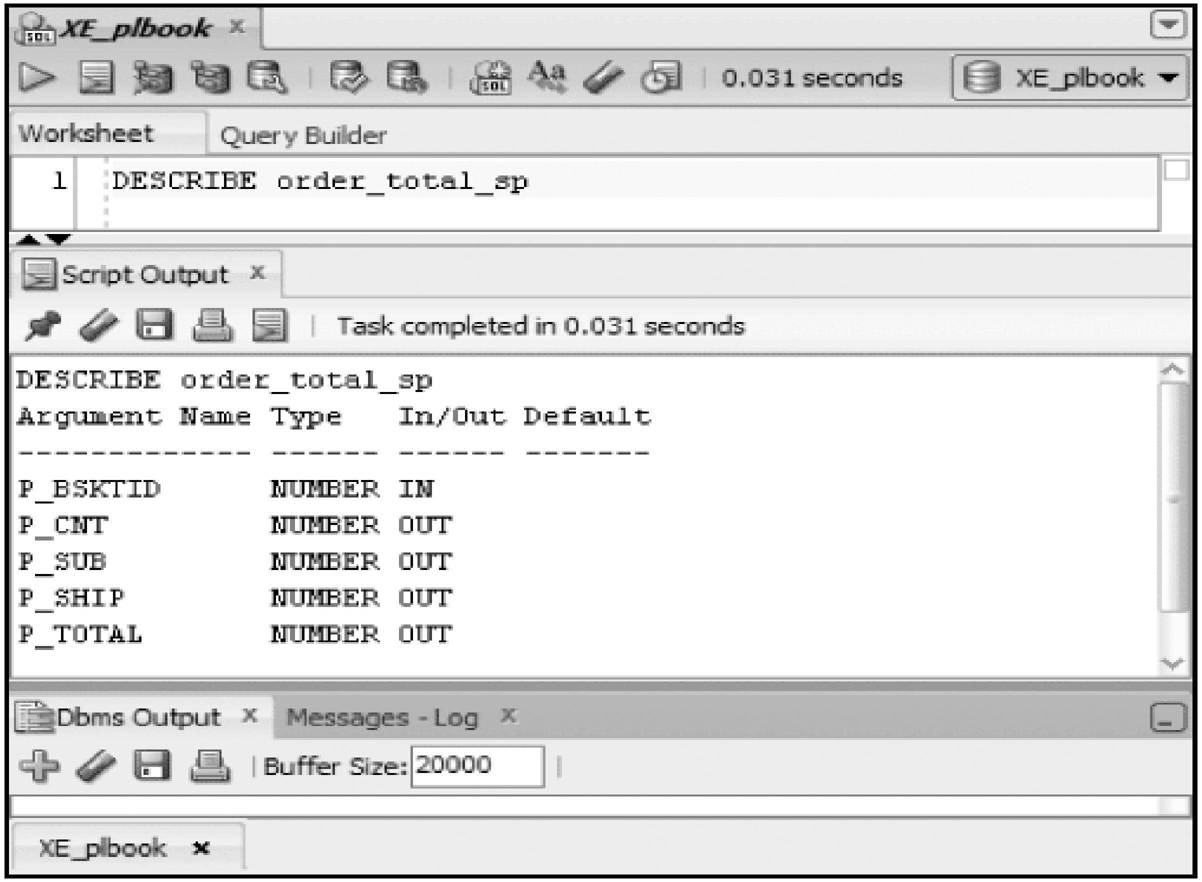
```
CREATE OR REPLACE PROCEDURE ORDER_TOTAL_SP
  (p_bsktid IN bb_basketitem.idbasket%TYPE,
   p_cnt OUT NUMBER,
   p_sub OUT NUMBER,
   p_ship OUT NUMBER,
   p_total OUT NUMBER)
 IS
BEGIN
  DBMS_OUTPUT.PUT_LINE('order total proc called');
  SELECT SUM(quantity), SUM(quantity*price)
    INTO p_cnt, p_sub
    FROM bb_basketitem
    WHERE idbasket = p_bsktid;
  ship_cost_sp(p_cnt,p_ship);
  p_total := NVL(p_sub,0) + NVL(p_ship,0);
  DBMS_OUTPUT.PUT_LINE('order total proc ended');
END ORDER_TOTAL_SP;
```

Type this code

Call to the SHIP_COST_SP procedure

DESCRIBE Command

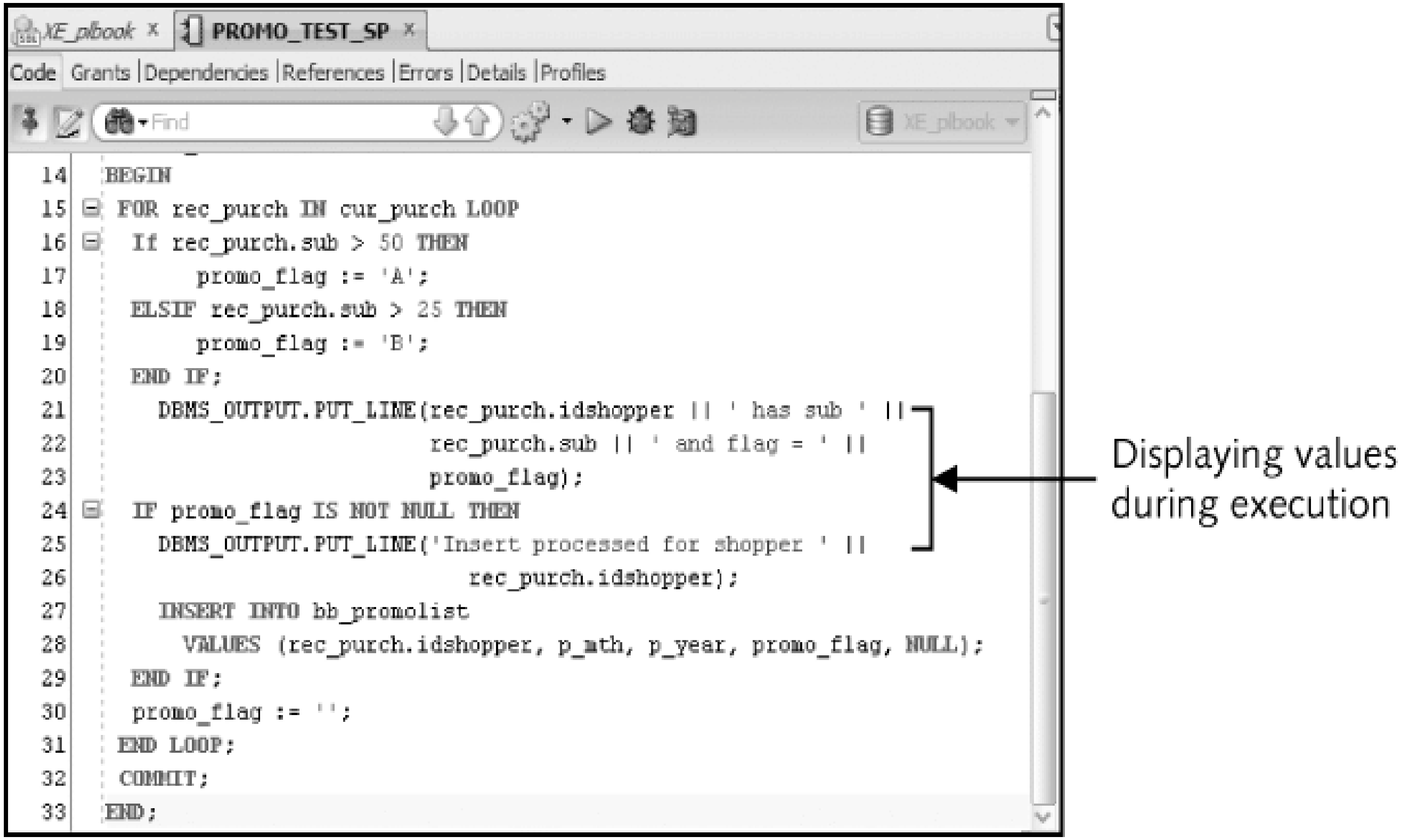
- Lists the parameters of a program unit



```
DESCRIBE order_total_sp
```

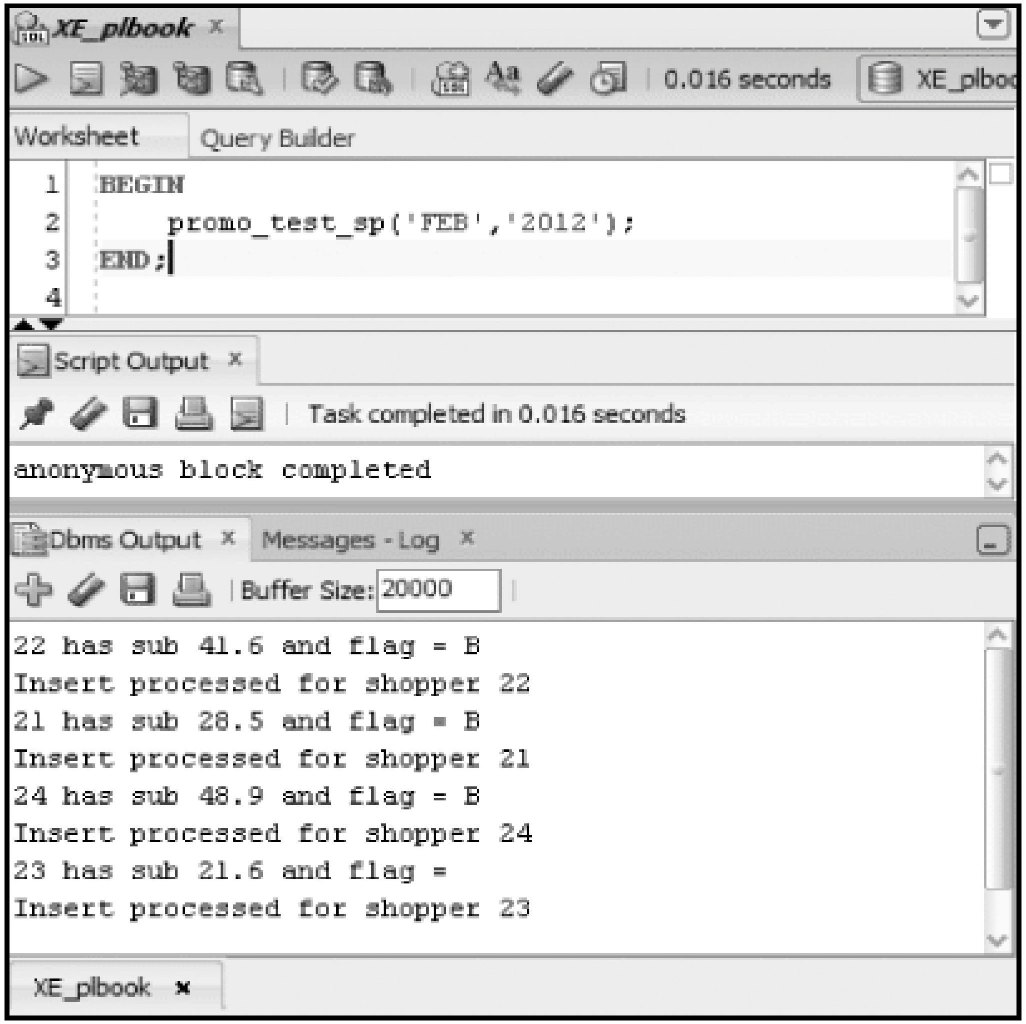
```
DESCRIBE order_total_sp
Argument Name Type   In/Out Default
------------- ------ ------ -------
P_BSKTID      NUMBER IN
P_CNT         NUMBER OUT
P_SUB         NUMBER OUT
P_SHIP        NUMBER OUT
P_TOTAL       NUMBER OUT
```

Debugging with DBMS_OUTPUT


XE_plbook
PROMO_TEST_SP
Code | Grants | Dependencies | References | Errors | Details | Profiles
Find
XE_plbook
Displaying values during execution

```
BEGIN
 FOR rec_purch IN cur_purch LOOP
  If rec_purch.sub > 50 THEN
       promo_flag := 'A';
  ELSIF rec_purch.sub > 25 THEN
       promo_flag := 'B';
  END IF;
    DBMS_OUTPUT.PUT_LINE(rec_purch.idshopper || ' has sub ' ||
                         rec_purch.sub || ' and flag = ' ||
                         promo_flag);
  IF promo_flag IS NOT NULL THEN
    DBMS_OUTPUT.PUT_LINE('Insert processed for shopper ' ||
                            rec_purch.idshopper);
    INSERT INTO bb_promolist
      VALUES (rec_purch.idshopper, p_mth, p_year, promo_flag, NULL);
  END IF;
  promo_flag := '';
 END LOOP;
 COMMIT;
END;
```

Debugging with DBMS_OUTPUT



```
BEGIN
    promo_test_sp('FEB','2012');
END;
```

```
anonymous block completed
```

```
22 has sub 41.6 and flag = B
Insert processed for shopper 22
21 has sub 28.5 and flag = B
Insert processed for shopper 21
24 has sub 48.9 and flag = B
Insert processed for shopper 24
23 has sub 21.6 and flag =
Insert processed for shopper 23
```

Subprograms

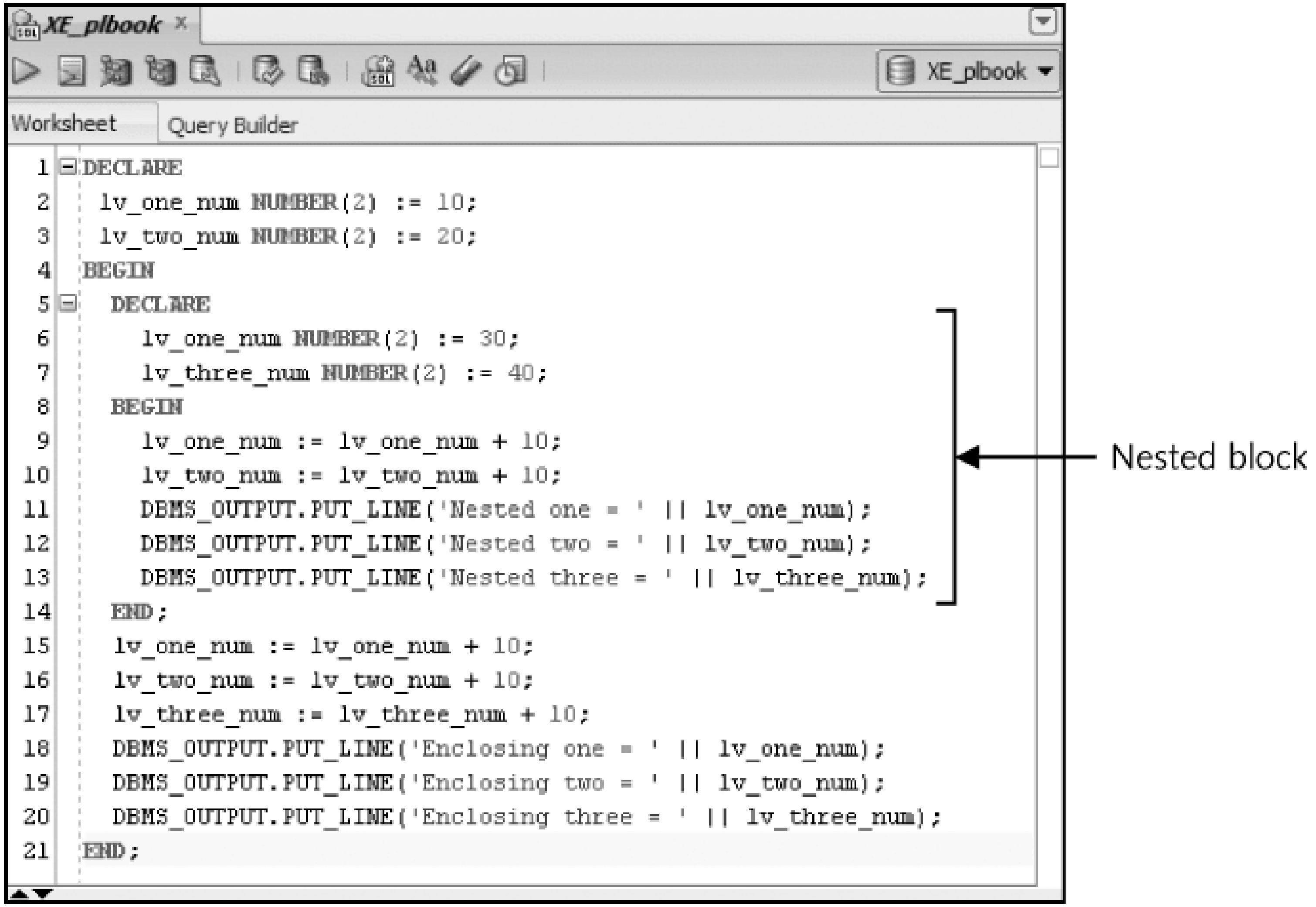
- A program unit defined within another program unit
- Must be declared in the DECLARE section of the containing program unit
- Can only be referenced by the containing program unit

P
L
/
S
Q
L

Variable Scope

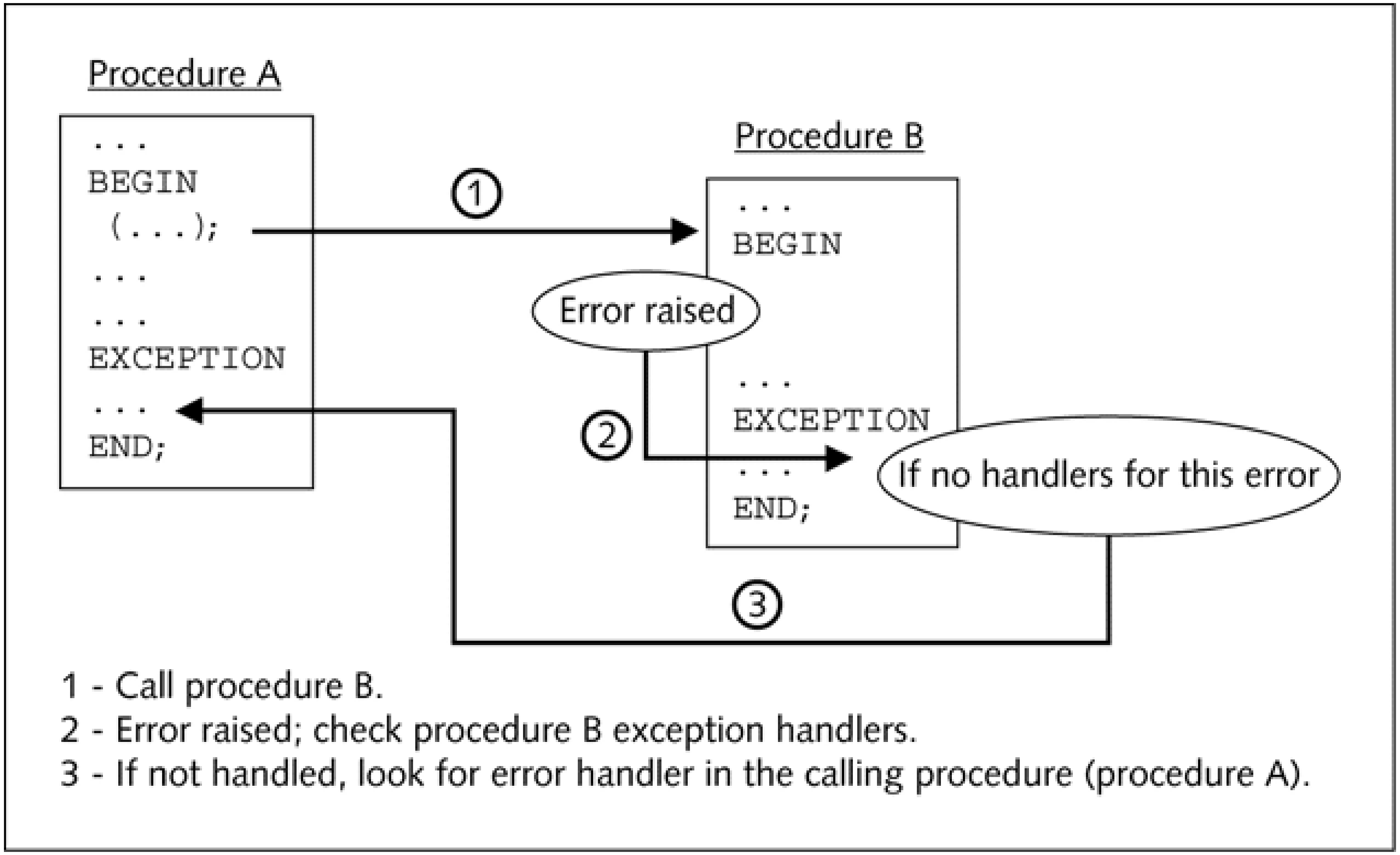
- When nesting blocks, are variables shared?
- Inner blocks can use variables from outer blocks

Variable Scope (continued)



```
DECLARE
 lv_one_num NUMBER(2) := 10;
 lv_two_num NUMBER(2) := 20;
BEGIN
  DECLARE
    lv_one_num NUMBER(2) := 30;
    lv_three_num NUMBER(2) := 40;
  BEGIN
    lv_one_num := lv_one_num + 10;
    lv_two_num := lv_two_num + 10;
    DBMS_OUTPUT.PUT_LINE('Nested one = ' || lv_one_num);
    DBMS_OUTPUT.PUT_LINE('Nested two = ' || lv_two_num);
    DBMS_OUTPUT.PUT_LINE('Nested three = ' || lv_three_num);
  END;
  lv_one_num := lv_one_num + 10;
  lv_two_num := lv_two_num + 10;
  lv_three_num := lv_three_num + 10;
  DBMS_OUTPUT.PUT_LINE('Enclosing one = ' || lv_one_num);
  DBMS_OUTPUT.PUT_LINE('Enclosing two = ' || lv_two_num);
  DBMS_OUTPUT.PUT_LINE('Enclosing three = ' || lv_three_num);
END;
```

Exception-HandlingFlow



1 - Call procedure B.
2 - Error raised; check procedure B exception handlers.
3 - If not handled, look for error handler in the calling procedure (procedure A).

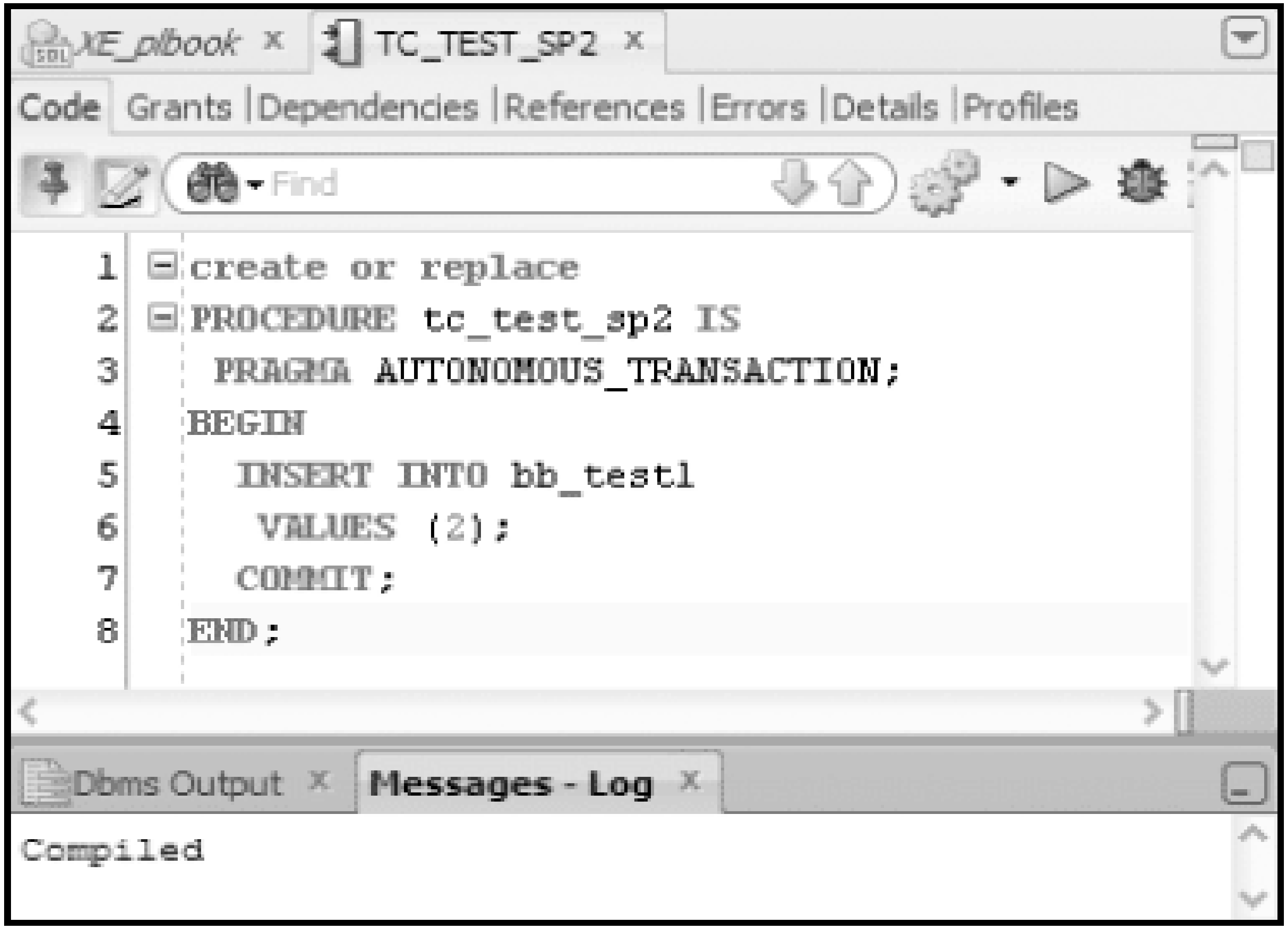
Transaction Control Scope

P
L
/
S
Q
L

- The scope refers to the group of DML statements that are affected by a particular transaction control statement
- By default, a session has a single DML queue and a transaction control statement would affect all DML in the queue regardless of which program unit initiated the statement
- DML statements of a program unit can be treated separately or as an autonomous transaction

P
L
/
S
Q
L

Autonomous Transaction

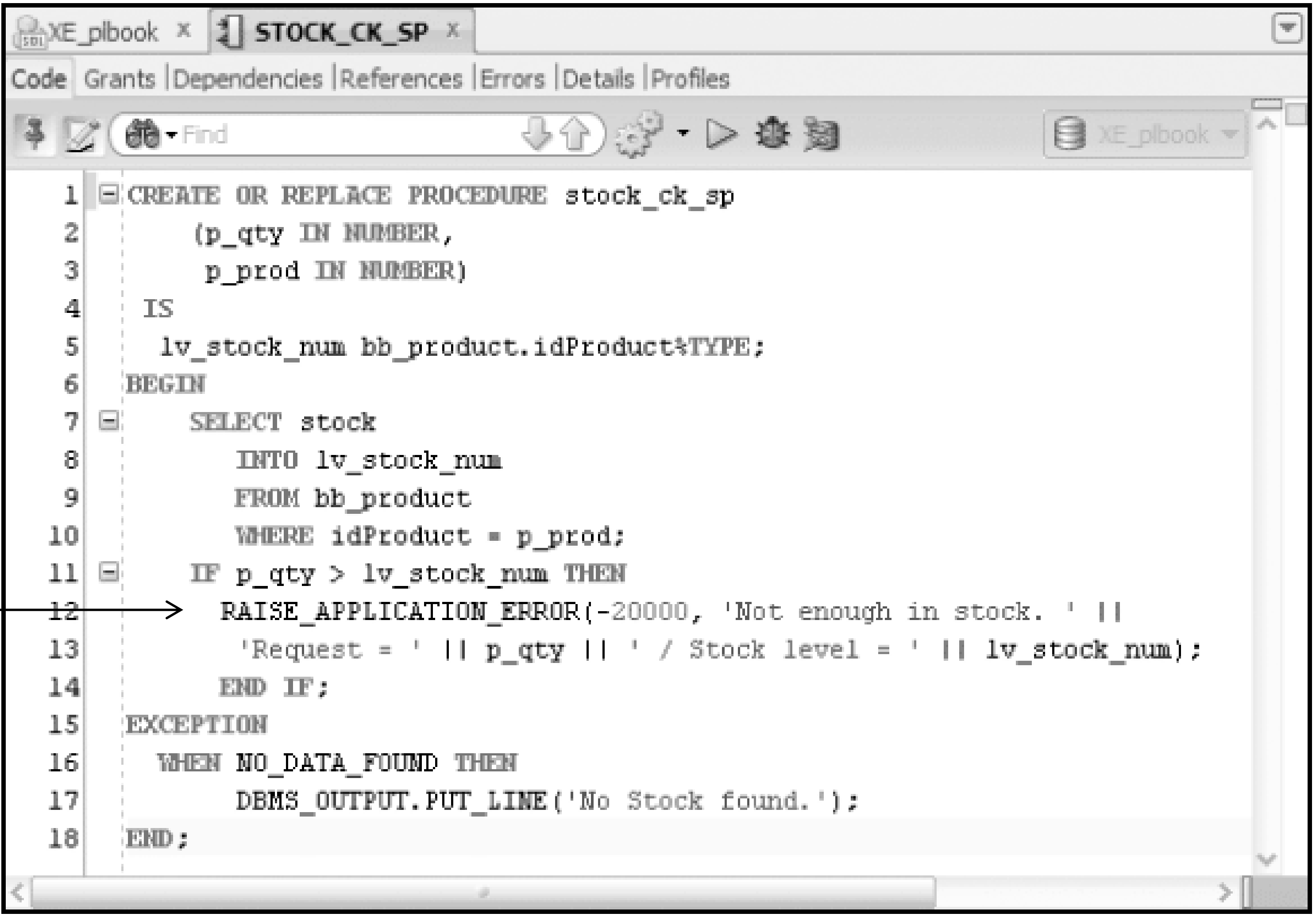


```
create or replace
PROCEDURE tc_test_sp2 IS
 PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
  INSERT INTO bb_test1
   VALUES (2);
  COMMIT;
END;
```

```
Compiled
```

RAISE_APPLICATION_ERROR

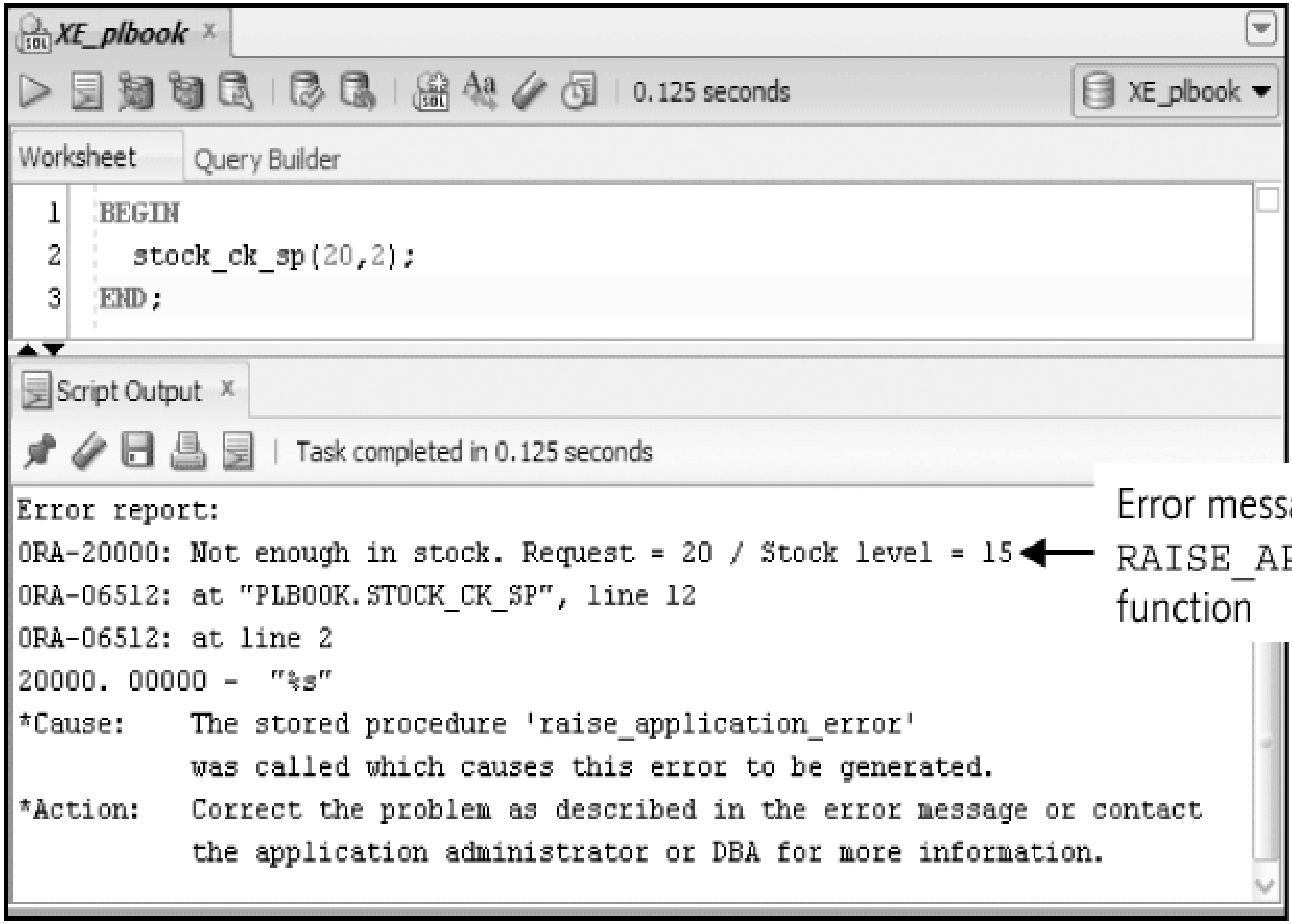
P
L
/
S
Q
L


XE_plbook
STOCK_CK_SP
Code | Grants | Dependencies | References | Errors | Details | Profiles
Find
XE_plbook

```
CREATE OR REPLACE PROCEDURE stock_ck_sp
    (p_qty IN NUMBER,
     p_prod IN NUMBER)
 IS
  lv_stock_num bb_product.idProduct%TYPE;
BEGIN
    SELECT stock
       INTO lv_stock_num
       FROM bb_product
       WHERE idProduct = p_prod;
    IF p_qty > lv_stock_num THEN
      RAISE_APPLICATION_ERROR(-20000, 'Not enough in stock. ' ||
       'Request = ' || p_qty || ' / Stock level = ' || lv_stock_num);
      END IF;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
       DBMS_OUTPUT.PUT_LINE('No Stock found.');
END;
```

RAISE_APPLICATION_ERROR (continued)

P
L
/
S
Q
L



```
BEGIN
  stock_ck_sp(20,2);
END;
```

```
Error report:
ORA-20000: Not enough in stock. Request = 20 / Stock level = 15
ORA-06512: at "PLBOOK.STOCK_CK_SP", line 12
ORA-06512: at line 2
20000. 00000 -  "%s"
*Cause:    The stored procedure 'raise_application_error'
           was called which causes this error to be generated.
*Action:   Correct the problem as described in the error message or contact
           the application administrator or DBA for more information.
```

Error message defined with the `RAISE_APPLICATION_ERROR` function

Remove a Procedure

```
DROP PROCEDURE procedure_name;
```

Summary

P
L
/
S
Q
L

- Named program unit assigns a name to a program unit so it can be reused
- Parameters are used to pass values in and out of program units
- Stored program units are saved in the database
- Parameter modes include: IN, OUT, and IN OUT
- Use DBMS_OUTPUT.PUT_LINE statement to debug

Summary (continued)

- A subprogam is a procedure declared within another procedure
- Variable scope must be considered with nested blocks
- Autonomous transactions must be explicitly created
- The RAISE_APPLICATION_ERROR function enables programmer defined errors
- Remove a procedure with the DROP PROCEDURE command