P
L
/
S
Q
L

**Oracle11*g*:
PL/SQL Programming**

Chapter 3

Handling Data in PL/SQL Blocks

Chapter Objectives

P
L
/
S
Q
L

- 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)

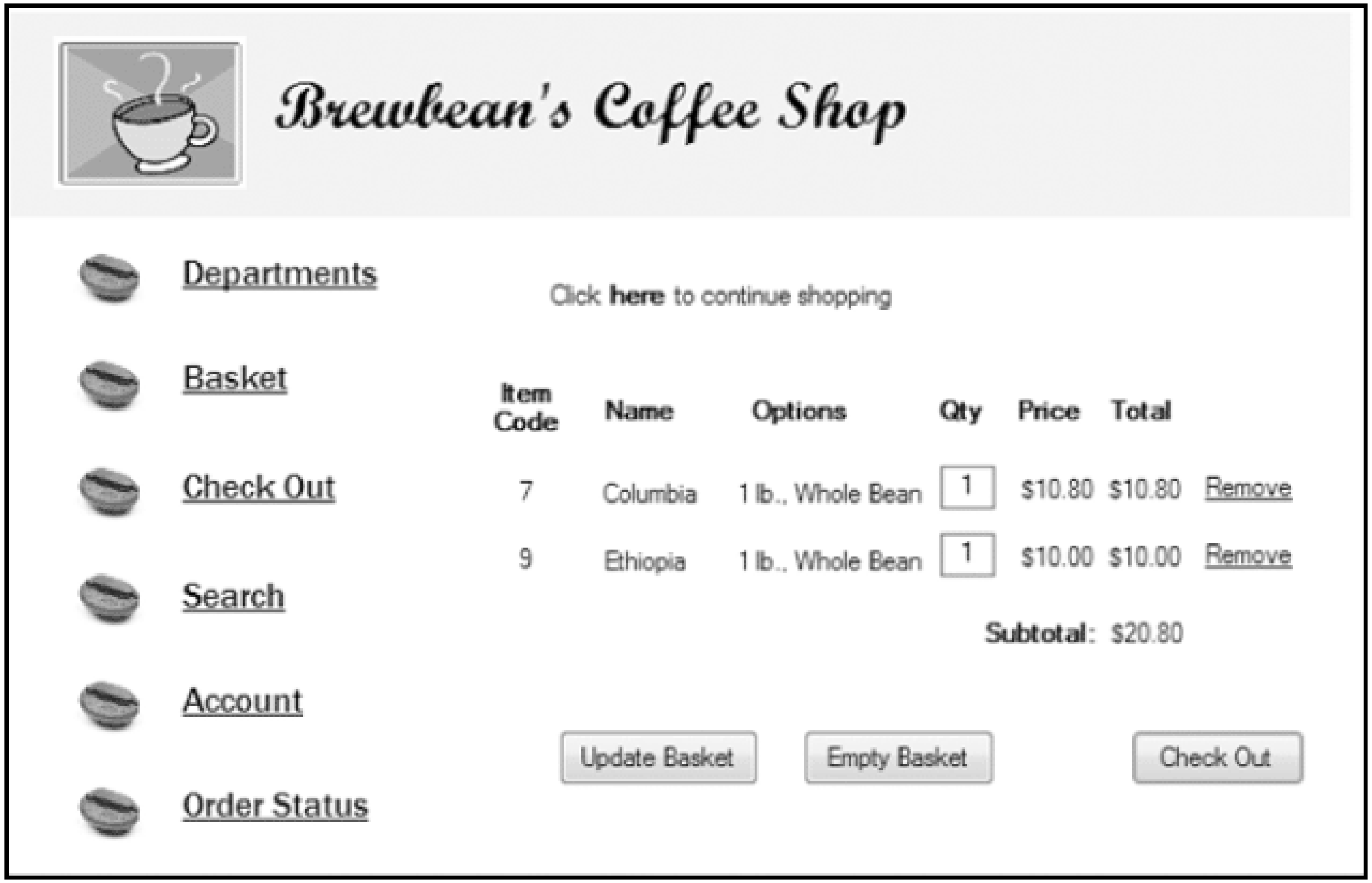
P
L
/
S
Q
L

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

Brewbean's Challenge

P L / S Q L

- Consider actions needed upon check out


Brewbean's Coffee Shop

Click **here** to continue shopping

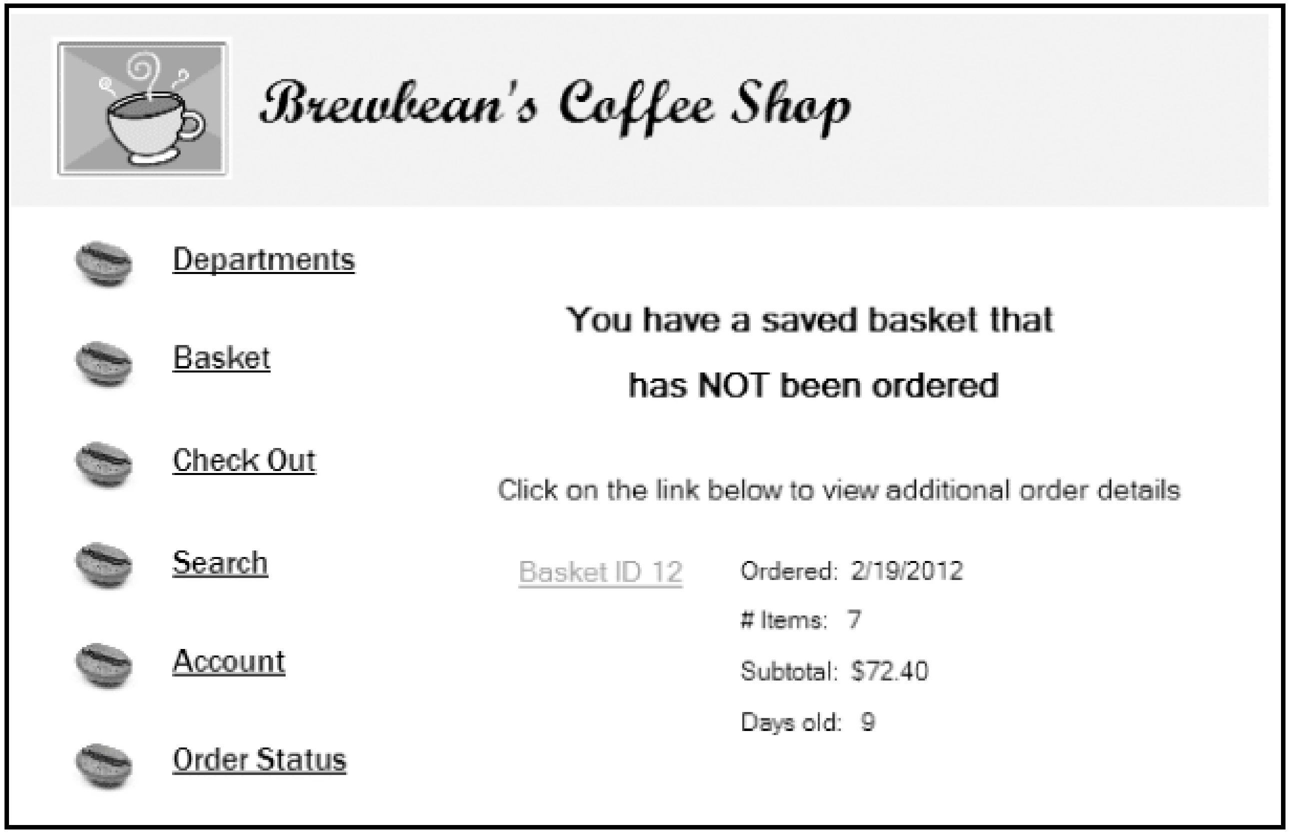
Item Code	Name	Options	Qty	Price	Total	
7	Columbia	1 lb., Whole Bean	1	$10.80	$10.80	Remove
9	Ethiopia	1 lb., Whole Bean	1	$10.00	$10.00	Remove

Subtotal: $20.80

Include SQL within a Block

P
L
/
S
Q
L

- 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

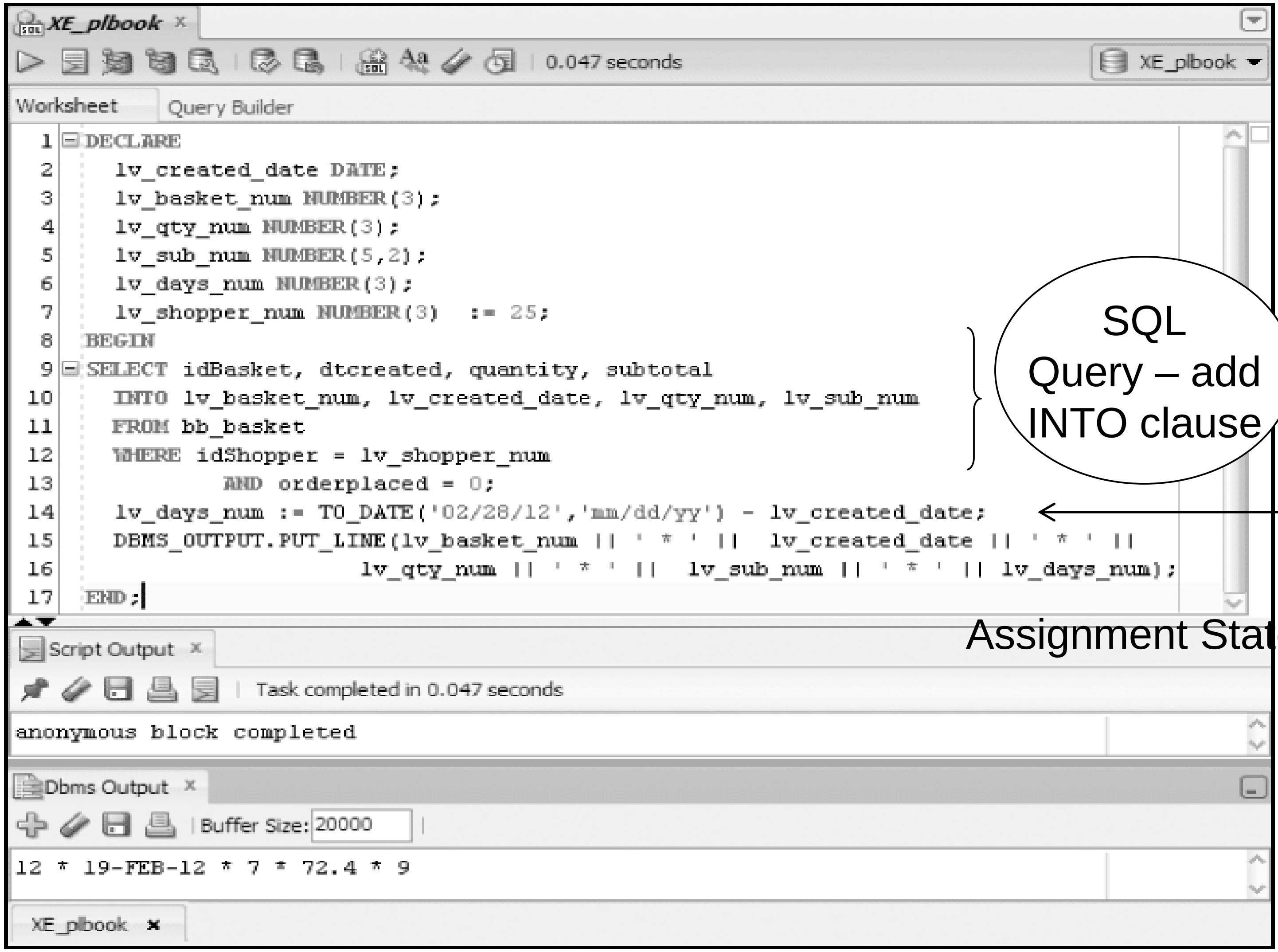
Include SQL within a Block (continued)

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

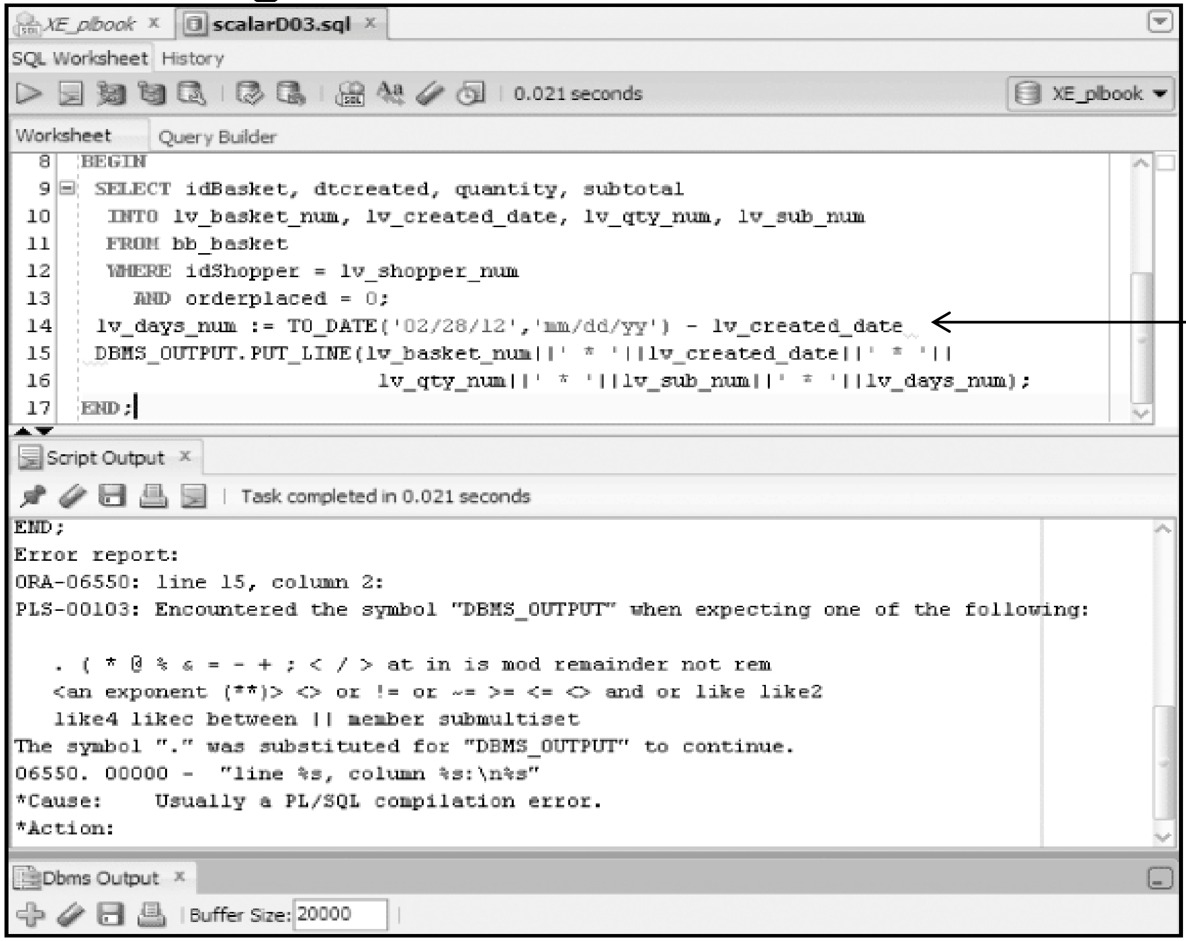
Assignment Statement

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:
```

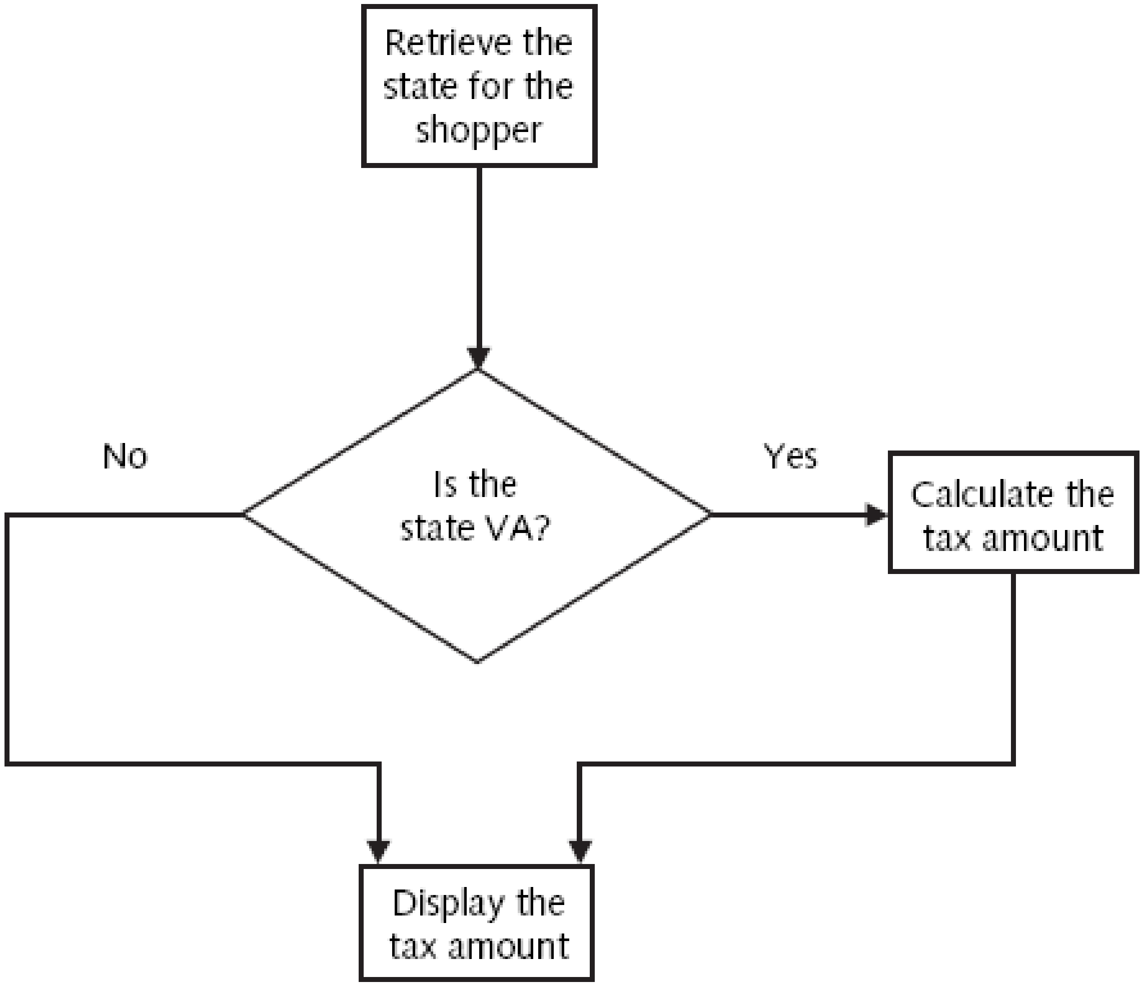
%TYPE Attribute

P
L
/
S
Q
L

- 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

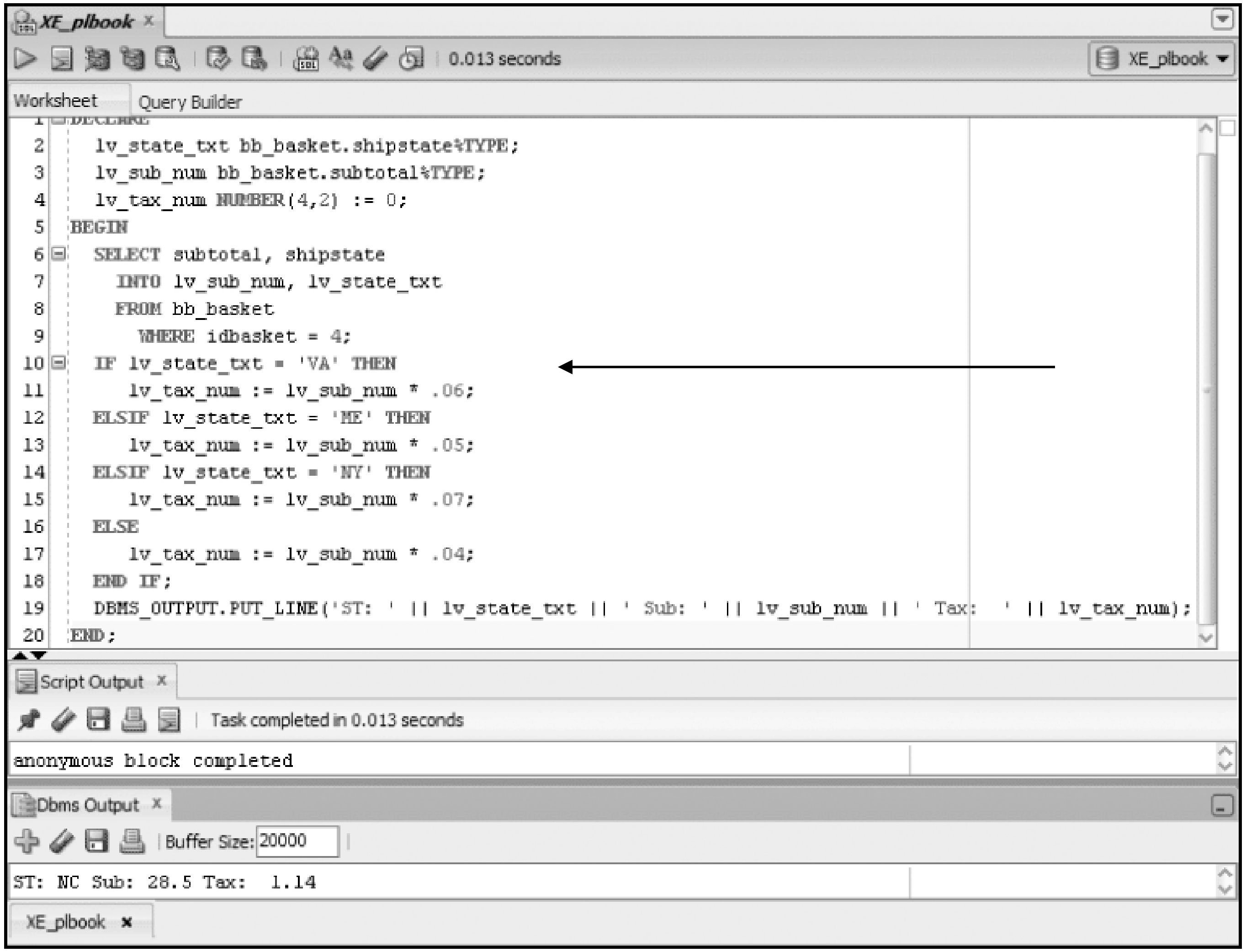
```
lv_basket_num bb_basket.idBasket%TYPE;
```

Data Retrieval with Decision Structures



IF Statement Example

P
L
/
S
Q
L



```
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;
```

```
anonymous block completed
```

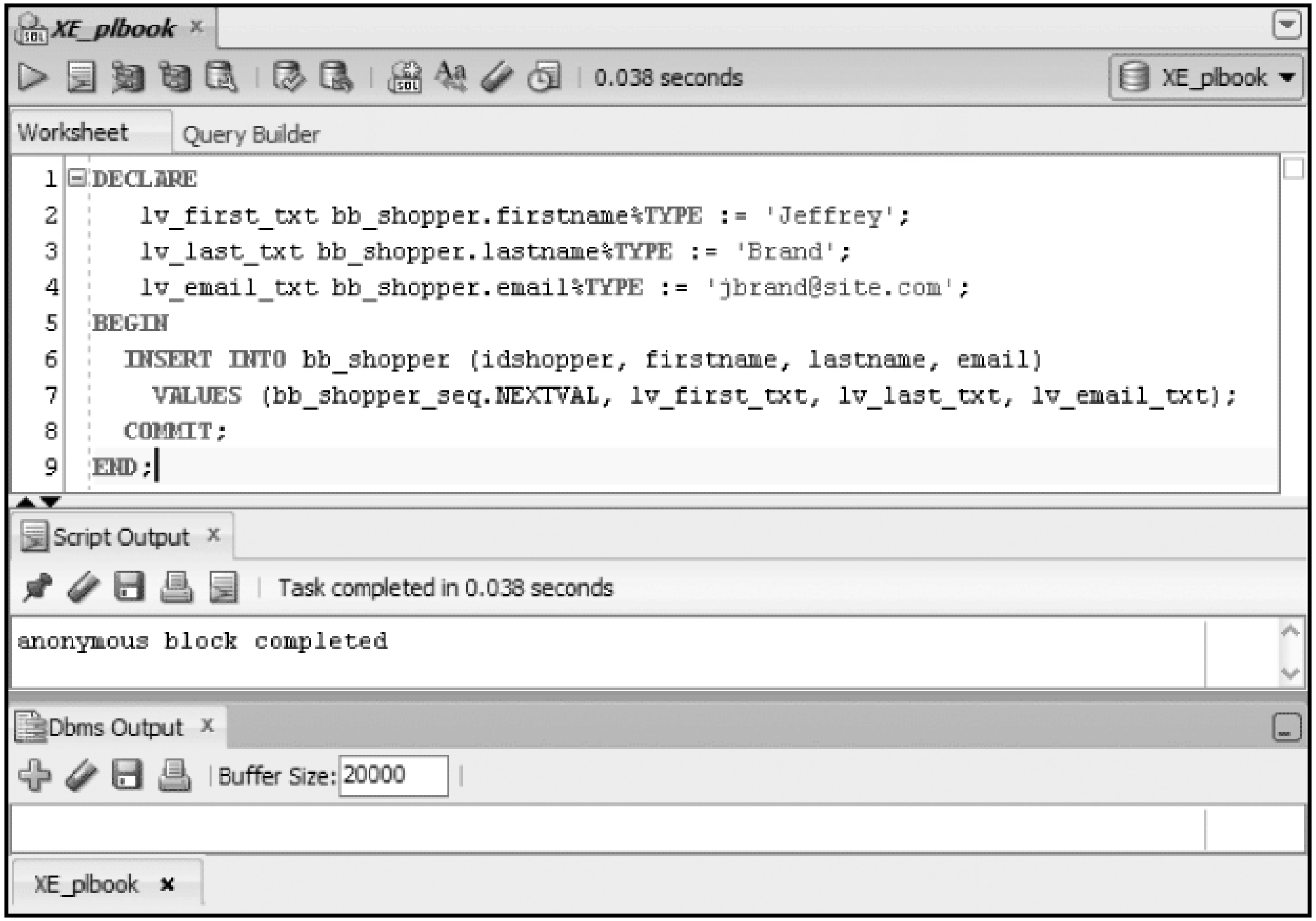
```
ST: NC Sub: 28.5 Tax:  1.14
```

Including DML

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

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;
```

Task completed in 0.038 seconds

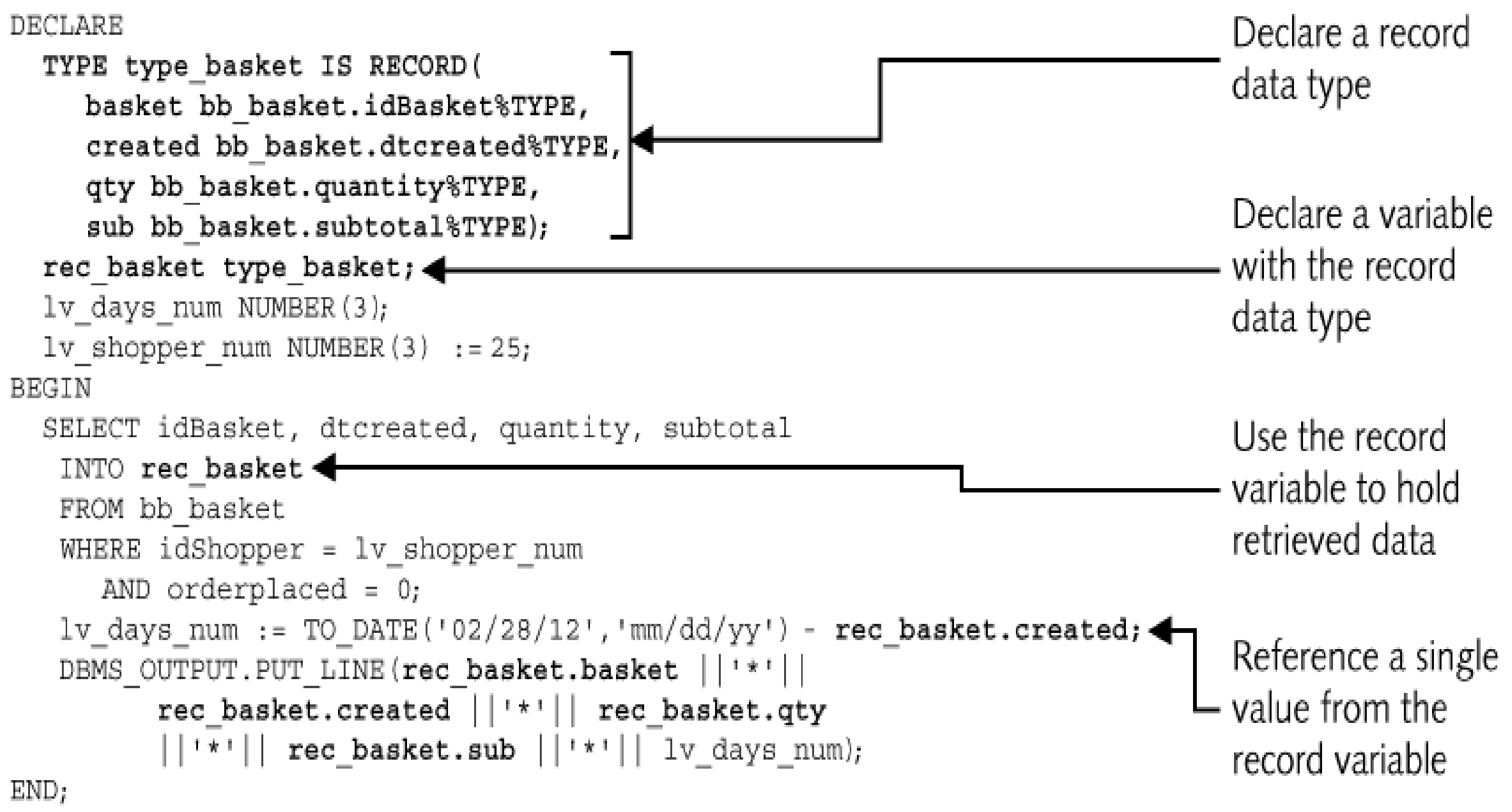
```
anonymous block completed
```

Record variables

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

Record Data Type

PL/SQL



```
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;
```

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

%ROWTYPE Attribute

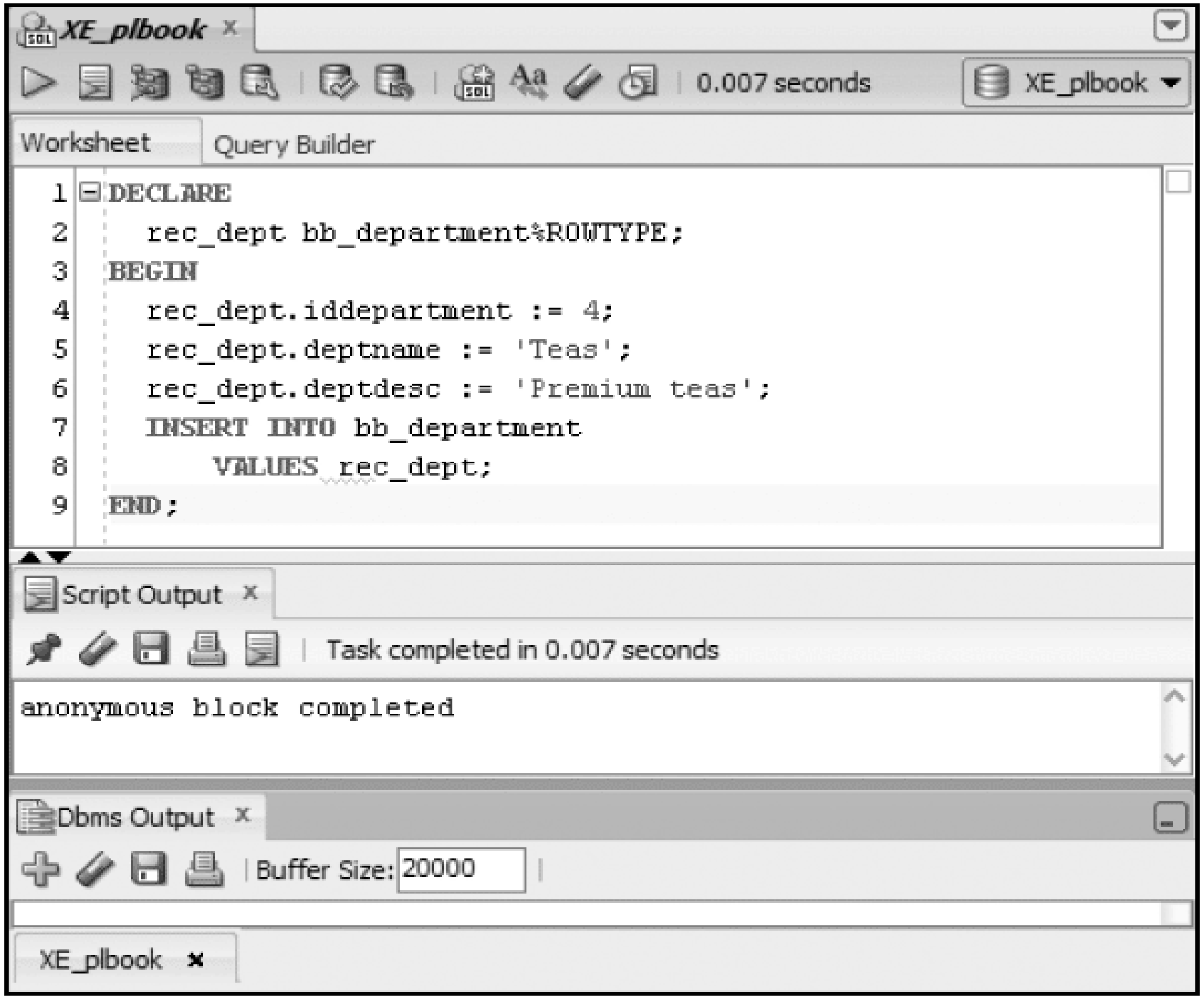
P
L
/
S
Q
L

- 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

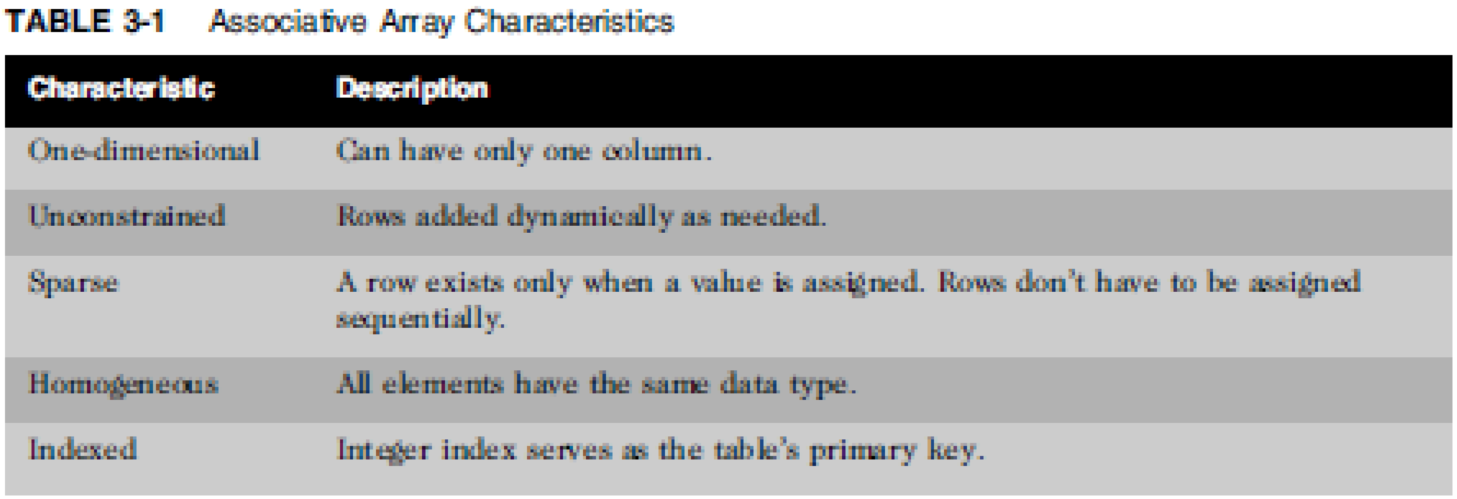


TABLE 3-1 Associative Array Characteristics

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

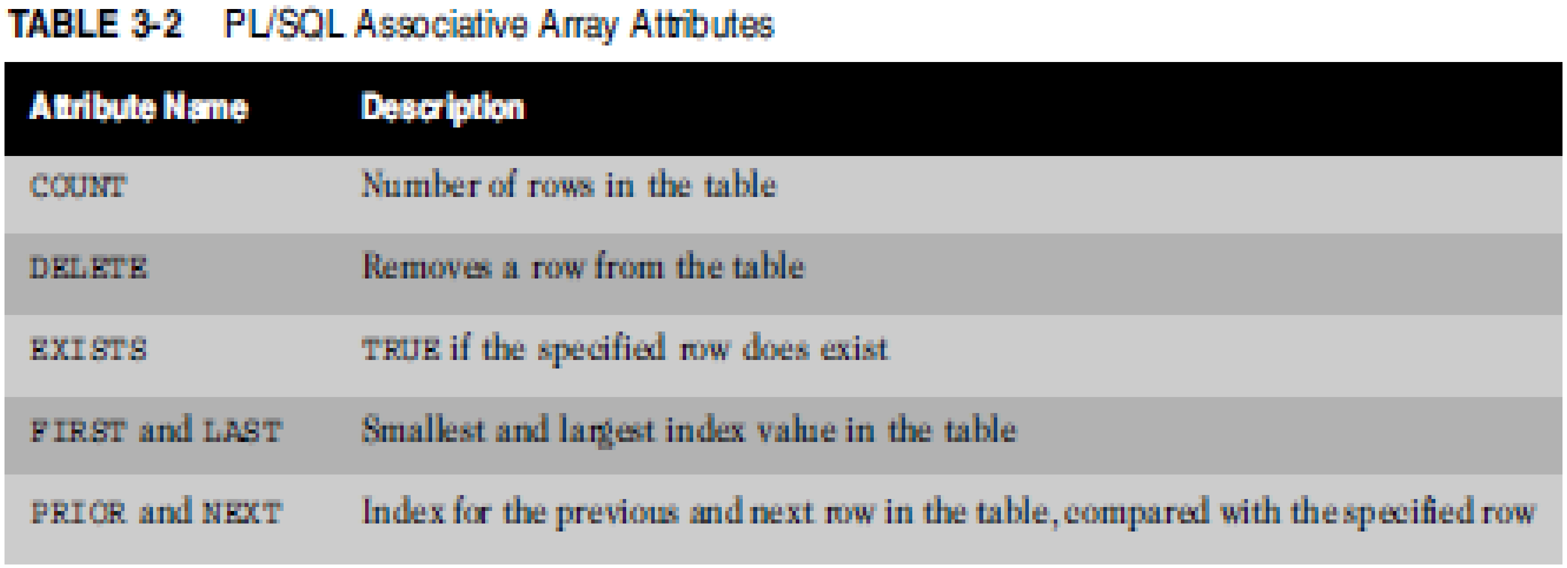
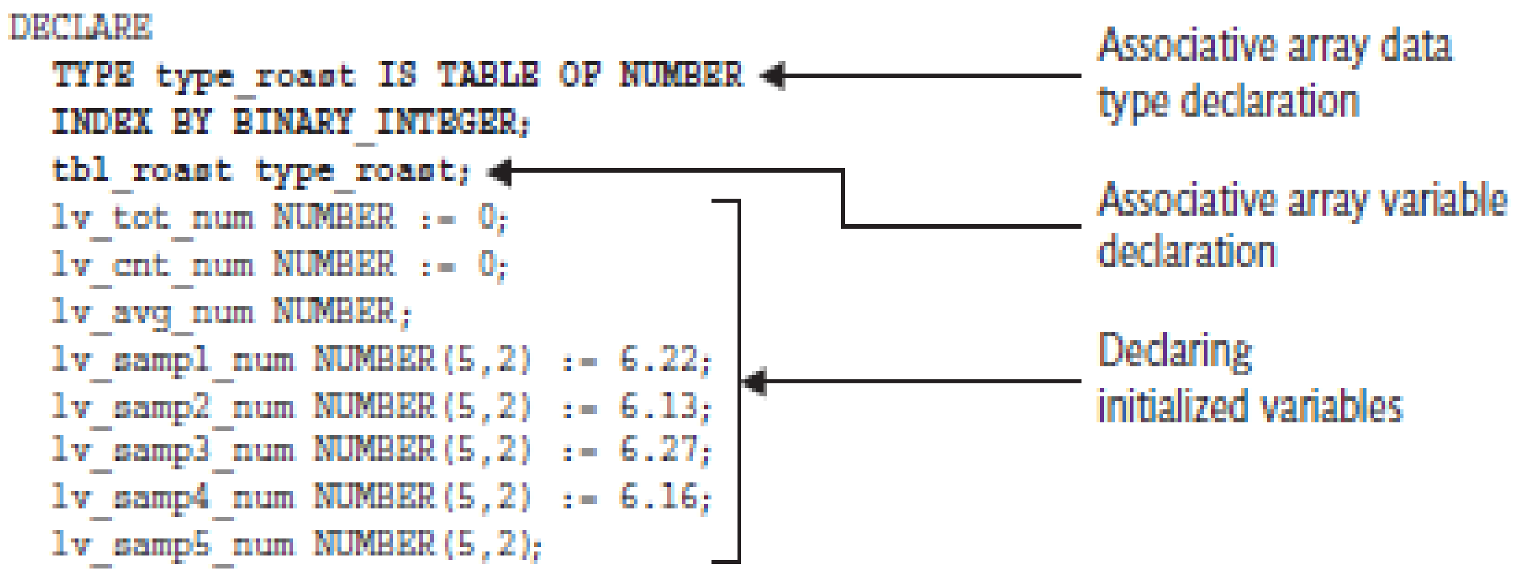


TABLE 3-2 PL/SQL Associative Array Attributes

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

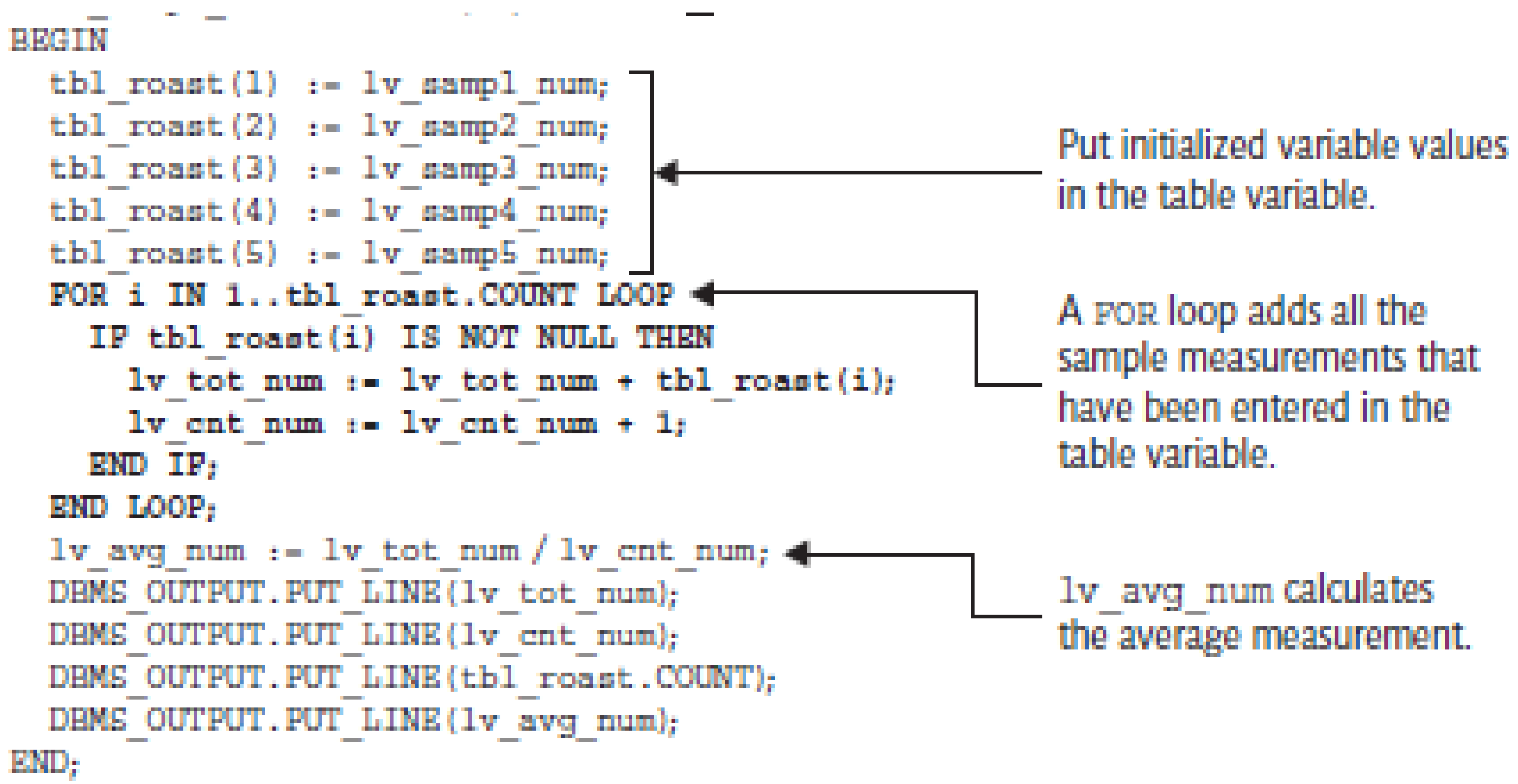
Associative Array Example



```
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);
```

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

Example (continued)



```
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;
```

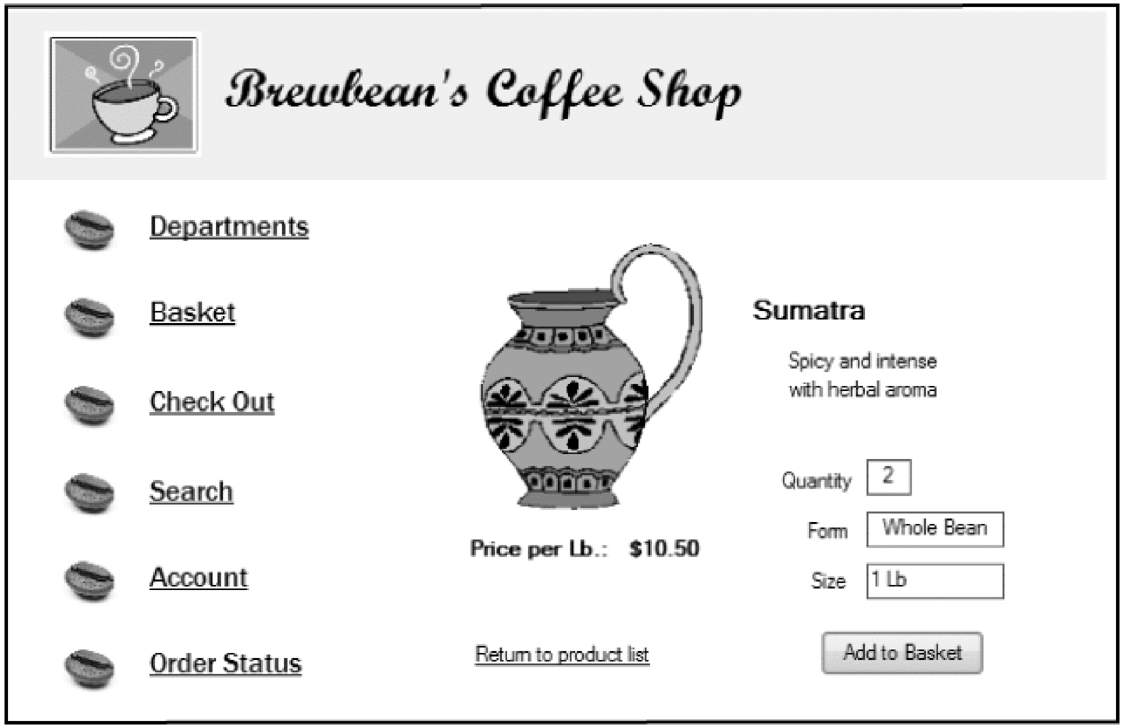
Put initialized variable values in the table variable.

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



Sumatra

Spicy and intense with herbal aroma

Price per Lb.: $10.50

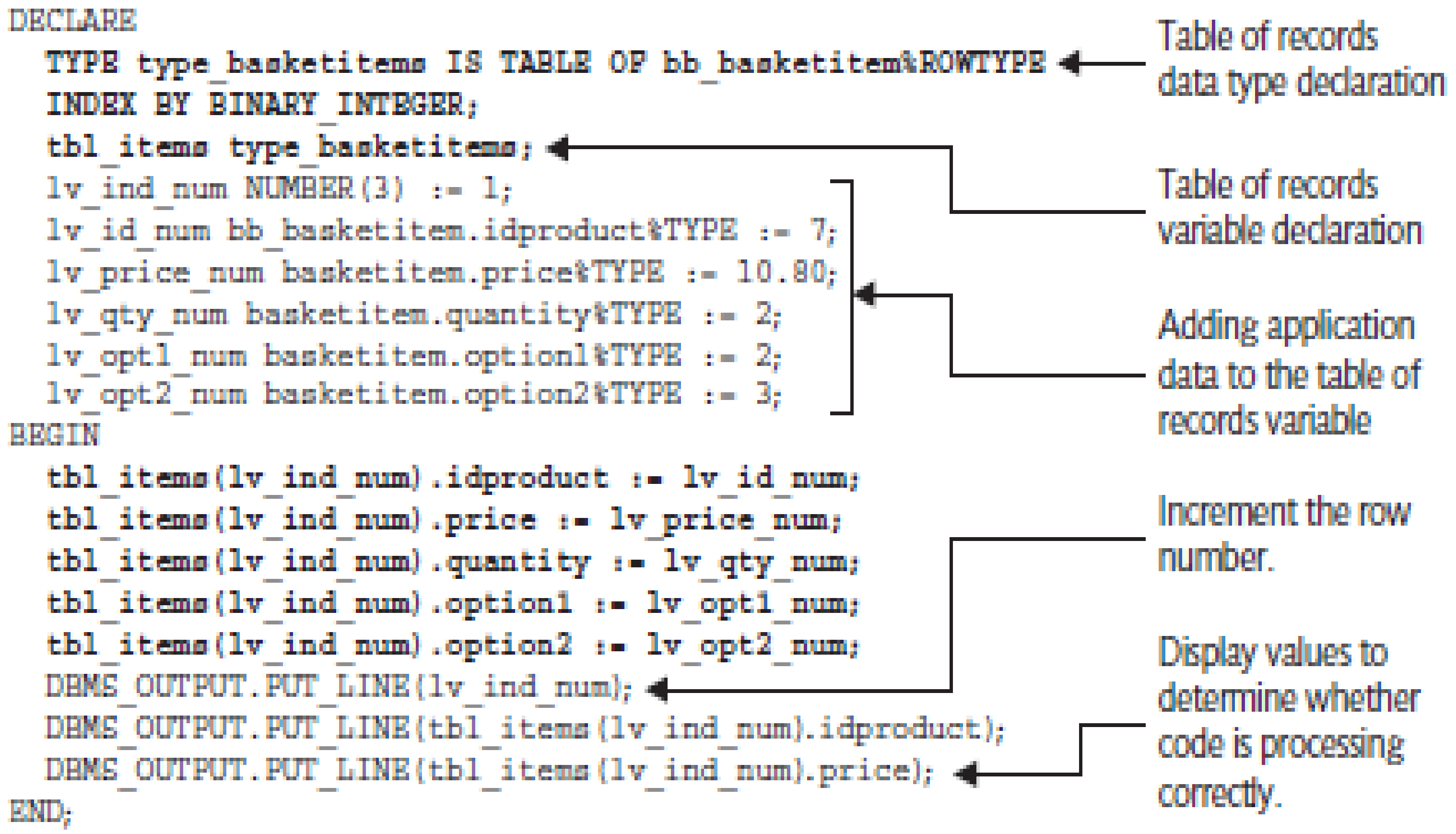
Quantity 2

Form Whole Bean

Size 1 Lb

Table of Records

P
L
/
S
Q
L



```
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.

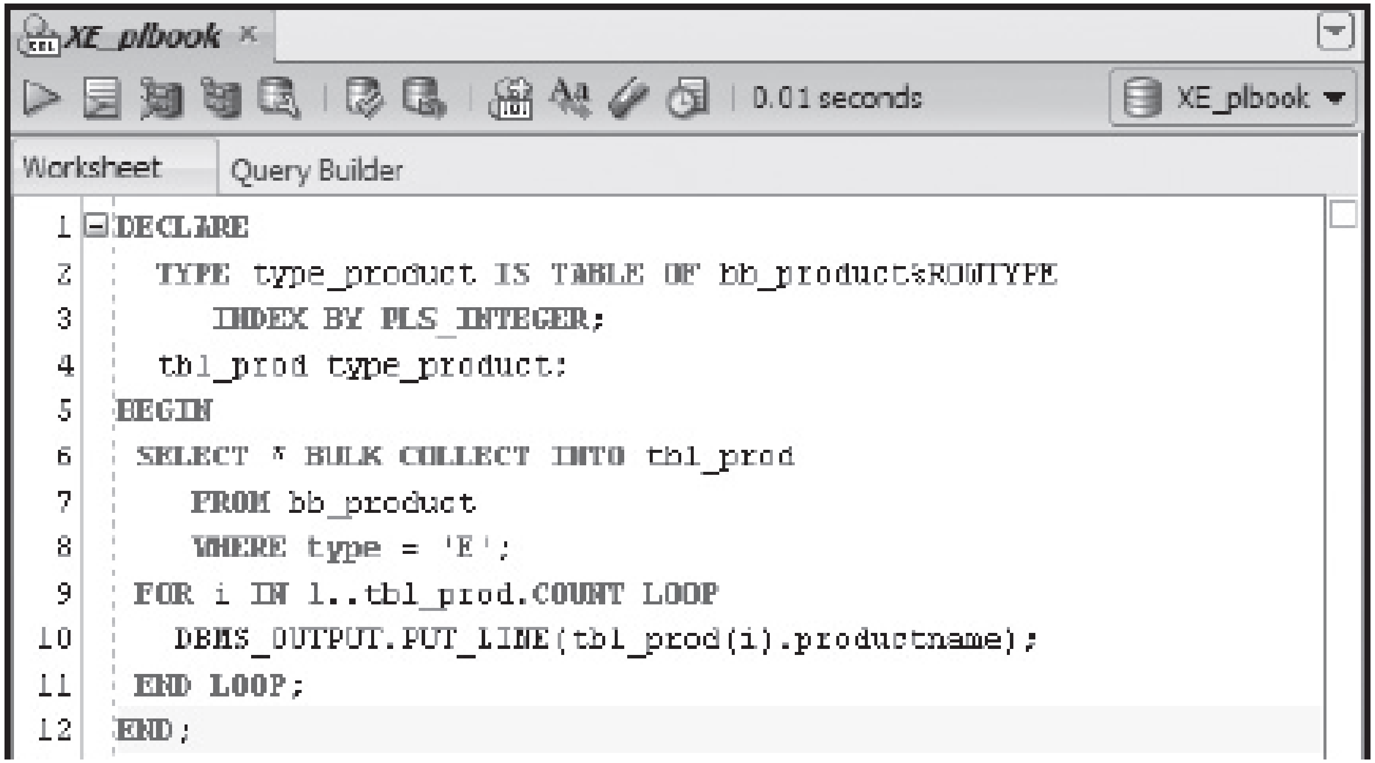
Bulk Processing

P
L
/
S
Q
L

- 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

P
L
/
S
Q
L

- 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

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