

Unit 3: Advanced SQL

Carlos Coronel, Steven Morris, and Peter Rob

Chapter 8 and Chapter 14

Unit 3: Advanced SQL

- ▶ Advanced SQL
 - ▶ ORACLE sequences.
 - ▶ Procedural SQL-
 - ▶ Anonymous PL/SQL blocks
 - ▶ Triggers,
 - ▶ Stored Procedures,
 - ▶ PL/SQL Cursors.
 - ▶ Embedded SQL – Dynamic SQL.
- ▶ EXTENSIBLE MARKUP LANGUAGE (XML)
 - ▶ DTD and XML Schemas,
 - ▶ XML presentation,
 - ▶ XML Applications

Table name: EMPLOYEE

EMP_NUM	EMP_TITLE	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_DOB	EMP_HIRE_DATE
100	Mr.	Kolmycz	George	D	15-Jun-42	15-Mar-88
101	Ms.	Lewis	Rhonda	G	19-Mar-65	25-Apr-86
102	Mr.	Vandam	Rhett		14-Nov-58	18-May-93
103	Ms.	Jones	Anne	M	11-May-74	26-Jul-99
104	Mr.	Lange	John	P	12-Jul-71	20-Aug-90
105	Mr.	Williams	Robert	D	14-Mar-75	19-Jun-03
106	Mrs.	Duzak	Jeanine	K	12-Feb-68	13-Mar-89
107	Mr.	Diante	Jorge	D	01-May-75	02-Jul-97
108	Mr.	Wiesenbach	Paul	R	14-Feb-66	03-Jun-93
109	Ms.	Travis	Elizabeth	K	18-Jun-61	14-Feb-06
110	Mrs.	Genkazi	Leighla	W	19-May-70	29-Jun-90

Table name: CUSTOMER

CUS_CODE	CUS_LNAME	CUS_FNAME	CUS_INITIAL	CUS_AREACODE	CUS_PHONE	CUS_BALANCE
10010	Ramas	Alfred	A	615	844-2573	0.00
10011	Dunne	Leona	K	713	894-1238	0.00
10012	Smith	Kathy	W	615	894-2285	896.54
10013	Olowski	Paul	F	615	894-2180	1285.19
10014	Orlando	Myron		615	222-1672	673.21
10015	O'Brian	Amy	B	713	442-3381	1014.56
10016	Brown	James	G	615	297-1228	0.00
10017	Williams	George		615	290-2556	0.00
10018	Farriss	Anne	G	713	382-7185	0.00
10019	Smith	Olette	K	615	297-3809	453.98

Table name: DEPENDENT

EMP_NUM	DEP_NUM	DEP_FNAME	DEP_DOB
1001	1	Annelise	05-Dec-97
1001	2	Jorge	30-Sep-02
1003	1	Suzanne	25-Jan-04
1006	1	Carlos	25-May-01
1008	1	Michael	19-Feb-95
1008	2	George	27-Jun-98
1008	3	Katherine	18-Aug-03

Table name: PRODUCT

PROD_CODE	PROD_DESCRIPTOR	PROD_PRICE	PROD_ON_HAND	VEND_CODE
001278-AB	Claw hammer	12.95	23	232
123-21UUY	Houselite chain saw, 16-in. bar	189.99	4	235
QER-34256	Sledge hammer, 16-lb. head	18.63	6	231
SRE-657UG	Rat-tail file	2.99	15	232
ZZX/3245Q	Steel tape, 12-ft. length	6.79	8	235

Table name: **VENDOR**

VEND_CODE	VEND_CONTACT	VEND_AREACODE	VEND_PHONE
230	Shelly K. Smithson	608	555-1234
231	James Johnson	615	123-4536
232	Annelise Crystall	608	224-2134
233	Candice Wallace	904	342-6567
234	Arthur Jones	615	123-3324
235	Henry Ortozo	615	899-3425

Table name: INVOICE

INV_NUMBER	CUS_CODE	INV_DATE
1001	10014	08-Mar-10
1002	10011	08-Mar-10
1003	10012	08-Mar-10
1004	10011	09-Mar-10

Table name: LINE

INV_NUMBER	LINE_NUMBER	PROD_CODE	LINE_UNITS	LINE_PRICE
1001	1	123-21UUY	1	189.99
1001	2	SRE-657UG	3	2.99
1002	1	QER-34256	2	18.63
1003	1	ZZX/3245Q	1	6.79
1003	2	SRE-657UG	1	2.99
1003	3	001278-AB	1	12.95
1004	1	001278-AB	1	12.95
1004	2	SRE-657UG	2	2.99

ORACLE SEQUENCES

- ▶ MS Access creates a column named *ID* with an *AutoNumber* data type.
- ▶ MS SQL Server uses the Identity column property
- ▶ Oracle does not support the *AutoNumber* data type or the Identity column property.
- ▶ Instead, you can use a “sequence” to assign values to a column on a table.

ORACLE SEQUENCES

- ▶ But an Oracle sequence is very different from the Access AutoNumber data type and deserves close scrutiny:
 - Oracle sequences are an independent object in the database. (Sequences are not a data type.)
 - Oracle sequences have a name and can be used anywhere a value is expected.
 - Oracle sequences are not tied to a table or a column.
 - Oracle sequences generate a numeric value that can be assigned to any column in any table.
 - The table attribute to which you assigned a value based on a sequence can be edited and modified.
 - An Oracle sequence can be created and deleted anytime.

ORACLE SEQUENCES

- ▶ The basic syntax to create a sequence in Oracle is:

```
CREATE SEQUENCE name [START WITH n] [INCREMENT BY n]  
[CACHE | NOCACHE]
```

where:

- ▶ *name* is the name of the sequence.
- ▶ *n* is an integer value that can be positive or negative.
- ▶ *START WITH* specifies the initial sequence value. (The default value is 1.)
- ▶ *INCREMENT BY* determines the value by which the sequence is incremented. (The default increment value is 1. The sequence increment can be positive or negative to enable you to create ascending or descending sequences.)
- ▶ The *CACHE* or *NOCACHE* clause indicates whether Oracle will preallocate sequence numbers in memory. (Oracle preallocates 20 values by default in case of *NOCACHE*.)

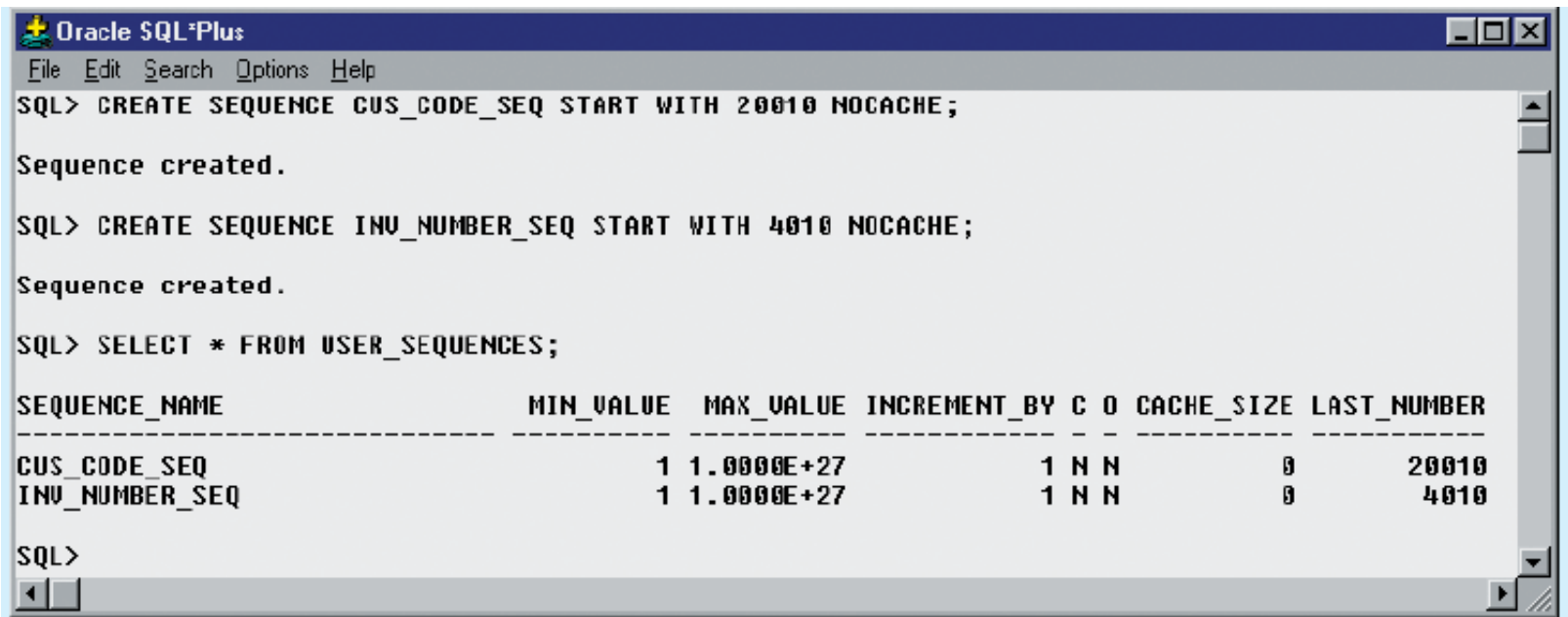
ORACLE SEQUENCES

► Example:

1. CREATE SEQUENCE CUS_CODE_SEQ START WITH 20010 NOCACHE;
2. CREATE SEQUENCE INV_NUMBER_SEQ START WITH 4010 NOCACHE;

► You can check all of the sequences you have created by using the following SQL command:

SELECT * FROM USER_SEQUENCES;



The screenshot shows an Oracle SQL*Plus window with the following text:

```
SQL> CREATE SEQUENCE CUS_CODE_SEQ START WITH 20010 NOCACHE;

Sequence created.

SQL> CREATE SEQUENCE INV_NUMBER_SEQ START WITH 4010 NOCACHE;

Sequence created.

SQL> SELECT * FROM USER_SEQUENCES;
```

SEQUENCE_NAME	MIN_VALUE	MAX_VALUE	INCREMENT_BY	C	O	CACHE_SIZE	LAST_NUMBER
CUS_CODE_SEQ	1	1.0000E+27	1	N	N	0	20010
INV_NUMBER_SEQ	1	1.0000E+27	1	N	N	0	4010

SQL>

ORACLE SEQUENCES

- ▶ To use sequences during data entry, you must use two special pseudo-columns: NEXTVAL and CURRVAL. NEXTVAL retrieves the next available value from a sequence, and CURRVAL retrieves the current value of a sequence.
- ▶ For example, you can use the following code to enter a new customer:

```
INSERT INTO CUSTOMER VALUES (CUS_CODE_SEQ.NEXTVAL, 'Connery',  
    'Sean', NULL, '615', '898-2008', 0.00);
```
- ▶ The preceding SQL statement adds a new customer to the CUSTOMER table and assigns the value 20010 to the CUS_CODE attribute.
- ▶ Some important sequence characteristics:
 - ▶ CUS_CODE_SEQ.NEXTVAL retrieves the next available value from the sequence.
 - ▶ Each time you use NEXTVAL, the sequence is incremented.
 - ▶ Once a sequence value is used (through NEXTVAL), it cannot be used again.
 - ▶ You can issue an INSERT statement without using the sequence.

```
Oracle SQL*Plus
File Edit Search Options Help
SQL> INSERT INTO CUSTOMER
  2  VALUES (CUS_CODE_SEQ.NEXTVAL, 'Connery', 'Sean', NULL, '615', '898-2007', 0.00);

1 row created.

SQL> SELECT * FROM CUSTOMER WHERE CUS_CODE = 20010;

  CUS_CODE CUS_LNAME      CUS_FNAME      C CUS CUS_PHON CUS_BALANCE
-----
    20010 Connery      Sean          615 898-2007          0

SQL> INSERT INTO INVOICE
  2  VALUES (INV_NUMBER_SEQ.NEXTVAL, 20010, SYSDATE);

1 row created.

SQL> SELECT * FROM INVOICE WHERE INV_NUMBER = 4010;

INV_NUMBER  CUS_CODE INV_DATE
-----
    4010      20010 04-MAY-09

SQL> INSERT INTO LINE
  2  VALUES (INV_NUMBER_SEQ.CURRVAL, 1, '13-Q2/P2', 1, 14.99);

1 row created.

SQL> INSERT INTO LINE
  2  VALUES (INV_NUMBER_SEQ.CURRVAL, 2, '23109-HB', 1, 9.95);

1 row created.

SQL> SELECT * FROM LINE WHERE INV_NUMBER = 4010;

INV_NUMBER LINE_NUMBER P_CODE      LINE_UNITS LINE_PRICE
-----
    4010          1 13-Q2/P2          1      14.99
    4010          2 23109-HB          1       9.95

SQL> COMMIT;

Commit complete.
```

ORACLE SEQUENCES

- ▶ The alternate syntax to create a sequence in Oracle is:

```
CREATE SEQUENCE sequence_name  
MINVALUE value  
MAXVALUE value  
START WITH value  
INCREMENT BY value  
CACHE/NOCACHE value  
CYCLE/NOCYCLE;
```

- ▶ Example:

```
CREATE SEQUENCE supplier_seq  
MINVALUE 1  
START WITH 1  
INCREMENT BY 1  
CACHE 20  
NOCYCLE;
```

- ▶ <https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/CREATE-SEQUENCE.html#GUID-E9C78A8C-615A-4757-B2A8-5E6EFB130571>

ORACLE SEQUENCES

- ▶ You can alter a sequence from a database with a `ALTER SEQUENCE` command. For example:

```
ALTER SEQUENCE CUS_CODE_SEQ  
INCREMENT BY 2;
```

- ▶ You can drop a sequence from a database with a `DROP SEQUENCE` command. For example:

```
DROP SEQUENCE CUS_CODE_SEQ;  
DROP SEQUENCE INV_NUMBER_SEQ;
```

- ▶ Dropping a sequence does not delete the values you assigned to table attributes (`CUS_CODE` and `INV_NUMBER`); it deletes only the sequence object from the database.
- ▶ The values you assigned to the table columns (*`CUS_CODE` and `INV_NUMBER`*) remain in the database.

Procedural SQL

- ▶ Procedural SQL-
 - ▶ Anonymous PL/SQL blocks
 - ▶ Triggers,
 - ▶ Stored Procedures,
 - ▶ PL/SQL Cursors.

Procedural SQL

- ▶ SQL does not support conditional execution of procedures.
- ▶ SQL also fails to support the looping operations
- ▶ Isolate critical code
 - ▶ All applications access shared code
 - ▶ Better maintenance and logic control
- ▶ **Persistent stored module (PSM)** is a block of code containing:
 - ▶ Standard SQL statements
 - ▶ Procedural extensions
 - ▶ Stored and executed at the DBMS server

Procedural SQL

- ▶ **Procedural SQL (PL/SQL)** enables you to:
 - ▶ Store procedural code and SQL statements in database
 - ▶ Merge SQL and traditional programming constructs
- ▶ Procedural code executed by DBMS when invoked by end user
 - ▶ Anonymous PL/SQL blocks
 - ▶ Triggers
 - ▶ Stored procedures and
 - ▶ PL/SQL functions

Anonymous PL/SQL block

- ▶ Using Oracle SQL*Plus, you can write a PL/SQL code block by enclosing the commands inside BEGIN and END clauses.
- ▶ For example, the following PL/SQL block inserts a new row in the VENDOR table:

```
BEGIN  
INSERT INTO VENDOR  
VALUES (25678,'Microsoft Corp. ', 'Bill Gates','765','546-8484','WA','N');  
END;  
/
```

- ▶ The following anonymous PL/SQL block inserts a row in the VENDOR table and displays the message “New Vendor Added!”

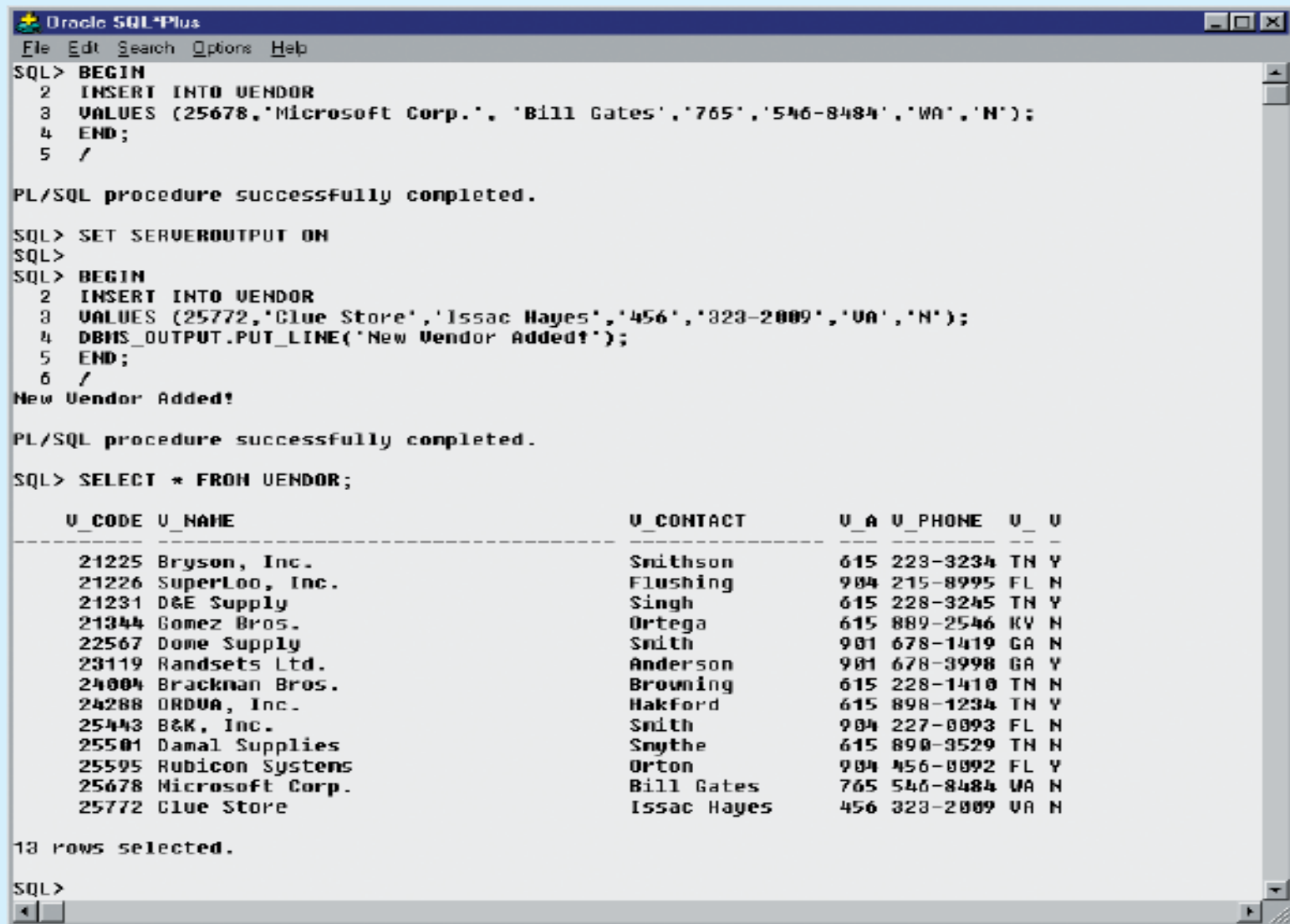
```
BEGIN  
INSERT INTO VENDOR  
VALUES (25772,'Clue Store', 'Issac Hayes', '456','323-2009', 'VA', 'N');  
DBMS_OUTPUT.PUT_LINE('New Vendor Added!');  
END;
```

/



**FIGURE
8.28**

Anonymous PL/SQL block examples



```
Oracle SQL*Plus
File Edit Search Options Help
SQL> BEGIN
  2  INSERT INTO VENDOR
  3  VALUES (25678,'Microsoft Corp.', 'Bill Gates','765','546-8484','WA','N');
  4  END;
  5  /

PL/SQL procedure successfully completed.

SQL> SET SERVEROUTPUT ON
SQL>
SQL> BEGIN
  2  INSERT INTO VENDOR
  3  VALUES (25772,'Glue Store','Issac Hayes','456','323-2009','VA','N');
  4  DBMS_OUTPUT.PUT_LINE('New Vendor Added!');
  5  END;
  6  /
New Vendor Added!

PL/SQL procedure successfully completed.

SQL> SELECT * FROM VENDOR;
```

V_CODE	V_NAME	V_CONTACT	V_A	V_PHONE	V_	V
21225	Bryson, Inc.	Smithson	615	228-3234	TN	Y
21226	SuperLoa, Inc.	Flushing	904	215-8995	FL	N
21231	D&E Supply	Singh	615	228-3245	TN	Y
21344	Gomez Bros.	Ortega	615	889-2546	KV	N
22567	Dome Supply	Smith	901	678-1419	GA	N
23119	Randsets Ltd.	Anderson	901	678-3998	GA	Y
24004	Bracknan Bros.	Browning	615	228-1410	TN	N
24288	ORDUA, Inc.	Hakford	615	898-1234	TN	Y
25443	B&K, Inc.	Smith	904	227-8093	FL	N
25501	Damal Supplies	Snythe	615	890-3529	TN	N
25595	Rubicon Systems	Orton	904	456-8092	FL	Y
25678	Microsoft Corp.	Bill Gates	765	546-8484	WA	N
25772	Glue Store	Issac Hayes	456	323-2009	VA	N

```
13 rows selected.

SQL>
```

Anonymous PL/SQL block

- ▶ The PL/SQL block shown in Figure 8.28 is known as an **anonymous PL/SQL block** because it has not been given a specific name.
- ▶ A PL/SQL block is defined by keywords DECLARE, BEGIN, EXCEPTION and END.
- ▶ DECLARE -- (optional)
 -- Variables, cursors, user-defined exceptions
- BEGIN -- (mandatory)
 -- PL/SQL statements
- EXCEPTION -- (optional)
 -- Actions to perform when errors occur
- END -- (mandatory)

Anonymous PL/SQL block with variables and loops

- ▶ A WHILE loop is used. Note the syntax:

WHILE condition LOOP

PL/SQL statements;

END LOOP

- ▶ The SELECT statement uses the INTO keyword to assign the output of the query to a PL/SQL variable. If the SELECT statement returns more than one value, you will get an error.
- ▶ The string concatenation symbol “||” to display the output.
- ▶ Each statement inside the PL/SQL code must end with a semicolon “;”.
- ▶ PL/SQL blocks can contain only standard SQL data manipulation language (DML) commands such as SELECT, INSERT, UPDATE, and DELETE.
- ▶ The use of data definition language (DDL) commands is not directly supported in a PL/SQL block.

Anonymous PL/SQL block with variables and loops

- ▶ The PL/SQL block starts with the DECLARE section in which you declare the variable names, the data types, and, if desired, an initial value. Supported data types are shown in Table 8.8.

TABLE 8.8 PL/SQL Basic Data Types

DATA TYPE	DESCRIPTION
CHAR	Character values of a fixed length; for example: W_ZIP CHAR(5)
VARCHAR2	Variable length character values; for example: W_FNAME VARCHAR2(15)
NUMBER	Numeric values; for example: W_PRICE NUMBER(6,2)
DATE	Date values; for example: W_EMP_DOB DATE
%TYPE	Inherits the data type from a variable that you declared previously or from an attribute of a database table; for example: W_PRICE PRODUCT.P_PRICE%TYPE Assigns W_PRICE the same data type as the P_PRICE column in the PRODUCT table

Anonymous PL/SQL block – Example 1

- ▶ The following example of an anonymous PL/SQL block demonstrates several of the constructs supported by the procedural language

```
DECLARE
```

```
W_P1 NUMBER(3) := 0;
```

```
W_P2 NUMBER(3) := 10;
```

```
W_NUM NUMBER(2) := 0;
```

```
BEGIN
```

```
WHILE W_P2 < 300 LOOP
```

```
    SELECT COUNT(P_CODE) INTO W_NUM FROM PRODUCT
```

```
    WHERE P_PRICE BETWEEN W_P1 AND W_P2;
```

```
    DBMS_OUTPUT.PUT_LINE('There are ' || W_NUM || ' Products  
with      price between ' || W_P1 || ' and ' || W_P2);
```

```
    W_P1 := W_P2 + 1;
```

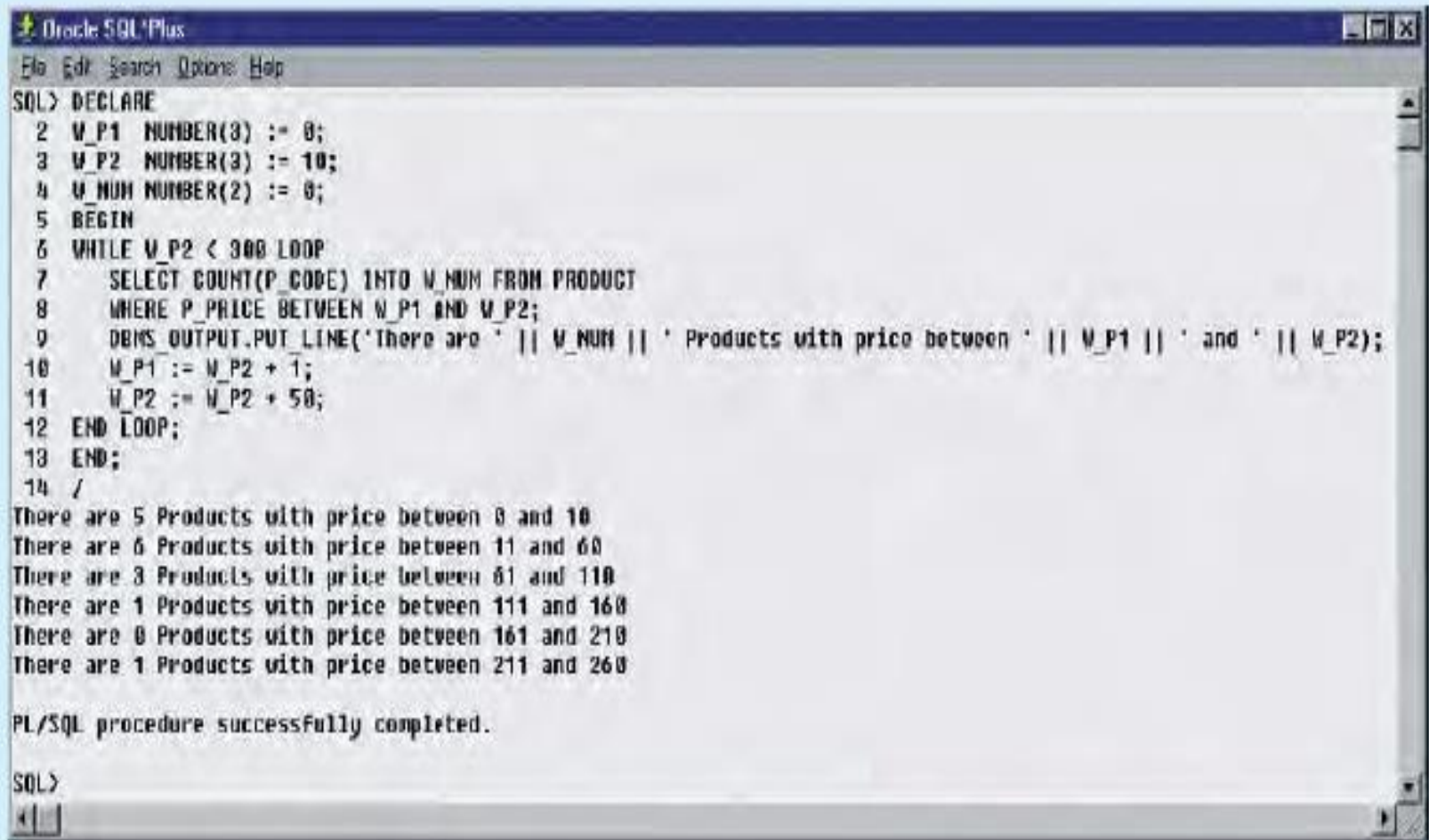
```
    W_P2 := W_P2 + 50;
```

```
END LOOP;
```

```
END;
```

**FIGURE
8.29**

Anonymous PL/SQL block with variables and loops



The screenshot shows a window titled "Oracle SQL*Plus" with a menu bar (File, Edit, Search, Options, Help). The main text area contains the following PL/SQL code and its output:

```
SQL> DECLARE
2  V_P1  NUMBER(3) := 0;
3  V_P2  NUMBER(3) := 10;
4  V_NUM NUMBER(2) := 0;
5  BEGIN
6  WHILE V_P2 < 300 LOOP
7      SELECT COUNT(P_CODE) INTO V_NUM FROM PRODUCT
8      WHERE P_PRICE BETWEEN V_P1 AND V_P2;
9      DBMS_OUTPUT.PUT_LINE('There are ' || V_NUM || ' Products with price between ' || V_P1 || ' and ' || V_P2);
10     V_P1 := V_P2 + 1;
11     V_P2 := V_P2 + 50;
12 END LOOP;
13 END;
14 /
```

There are 5 Products with price between 0 and 10
There are 6 Products with price between 11 and 60
There are 3 Products with price between 61 and 110
There are 1 Products with price between 111 and 160
There are 0 Products with price between 161 and 210
There are 1 Products with price between 211 and 260

PL/SQL procedure successfully completed.

SQL>

Anonymous PL/SQL block - Example 2

▶ Command> SET SERVEROUTPUT ON;

Command> DECLARE

 v_fname VARCHAR2 (20);

BEGIN

 SELECT first_name

 INTO v_fname

 FROM employees

 WHERE employee_id = 100;

 DBMS_OUTPUT.PUT_LINE (v_fname);

END;

/ Steven

PL/SQL procedure successfully completed.

Triggers

- ▶ A **trigger** is procedural SQL code that is automatically invoked by the RDBMS upon the occurrence of a given data manipulation event.
- ▶ It is useful to remember that:
 - ▶ A trigger is invoked before or after a data row is inserted, updated, or deleted.
 - ▶ A trigger is associated with a database table.
 - ▶ Each database table may have one or more triggers.
 - ▶ A trigger is executed as part of the transaction that triggered it.

Triggers

- ▶ Triggers are critical to proper database operation and management. For example:
 - ▶ Triggers can be used to enforce constraints that cannot be enforced at the DBMS design and implementation levels.
 - ▶ Triggers add functionality by automating critical actions and providing appropriate warnings and suggestions for remedial action.
 - ▶ In fact, one of the most common uses for triggers is to facilitate the enforcement of referential integrity.
 - ▶ Triggers can be used to update table values, insert records in tables, and call other stored procedures.

Triggers

- ▶ Oracle recommends triggers for:
 - ▶ Auditing purposes (creating audit logs).
 - ▶ Automatic generation of derived column values.
 - ▶ Enforcement of business or security constraints.
 - ▶ Creation of replica tables for backup purposes.

Triggers

```
CREATE OR REPLACE TRIGGER trigger_name
[BEFORE / AFTER] [DELETE / INSERT / UPDATE OF column_name]
  ON table_name
[FOR EACH ROW]
[DECLARE]
[variable_name data type[:=initial_value] ]
```

\\PL/SQL Instructions

```
BEGIN          -- (mandatory)
                -- PL/SQL statements
EXCEPTION -- (optional)
                -- Actions to perform when errors occur
END            -- (mandatory)
```

Triggers

- ▶ Procedural SQL code automatically invoked by RDBMS on data manipulation event
- ▶ **Trigger** definition:
 - ▶ Triggering timing: BEFORE or AFTER
 - ▶ Triggering event: INSERT, UPDATE, DELETE
 - ▶ Triggering level:
 - ▶ **Statement-level trigger** is assumed if you omit the FOR EACH ROW keywords. This type of trigger is executed once, before or after the triggering statement is completed. This is the default case
 - ▶ **Row-level trigger** requires use of the FOR EACH ROW keywords. This type of trigger is executed once for each row affected by the triggering statement.
 - ▶ Triggering action: The PL/SQL code enclosed between the BEGIN and END keywords. Each statement inside the PL/SQL code must end with a semicolon “;”
- ▶ DROP TRIGGER *trigger_name*

Triggers - Example

```
CREATE OR REPLACE TRIGGER t
BEFORE
    INSERT OR
    UPDATE OF salary, department_id OR
    DELETE
ON employees
BEGIN
CASE
    WHEN INSERTING THEN
        DBMS_OUTPUT.PUT_LINE('Inserting');
    WHEN UPDATING('salary') THEN
        DBMS_OUTPUT.PUT_LINE('Updating salary');
    WHEN UPDATING('department_id') THEN
        DBMS_OUTPUT.PUT_LINE('Updating department ID');
    WHEN DELETING THEN
        DBMS_OUTPUT.PUT_LINE('Deleting');
END CASE;
END;
/
```

**FIGURE
8.30**

The PRODUCT table

Oracle SQL*Plus

File Edit Search Options Help

SQL> SELECT * FROM PRODUCT;

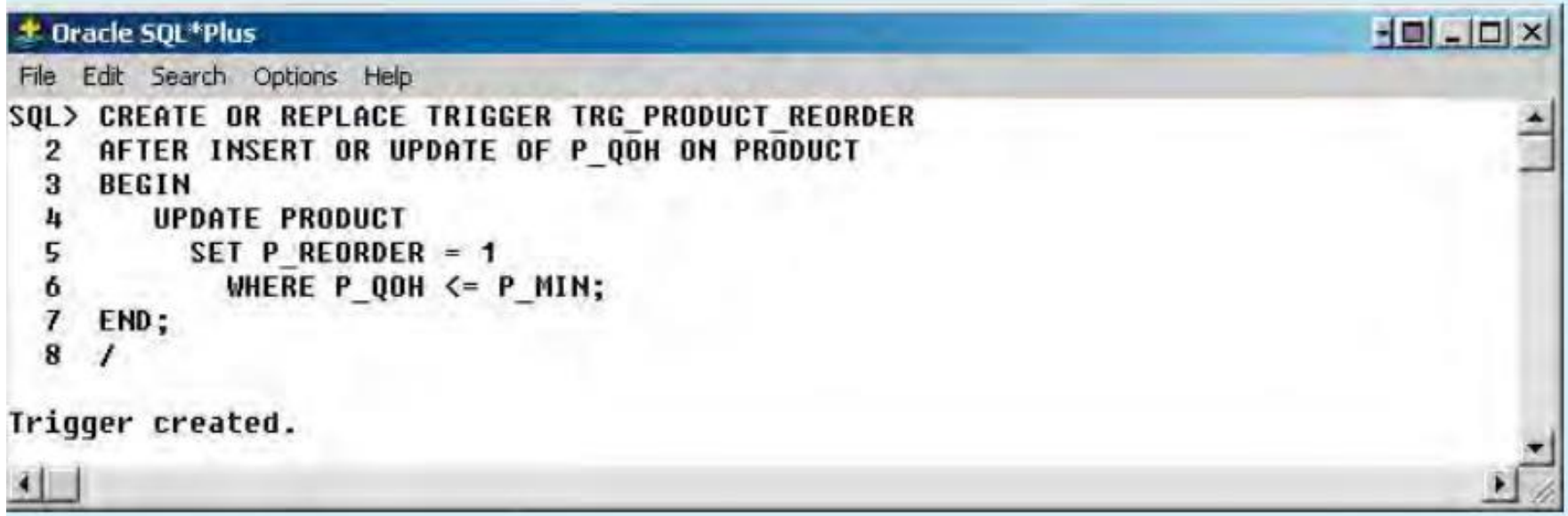
P_CODE	P_DESCRIPTION	P_INDATE	P_QOH	P_MIN	P_PRICE	P_DISCOUNT	U_CODE	P_MIN_ORDER	P_REORDER
11QER/31	Power painter, 15 psi., 3-nozz	03-NOV-09	8	5	189.99	0	25595	25	0
13-Q2/P2	7.25-in. pwr. saw blade	13-DEC-09	32	15	14.99	.05	21344	50	0
14-Q1/L3	9.00-in. pwr. saw blade	10-NOV-09	18	12	17.49	0	21344	50	0
1546-QQ2	Hrd. cloth, 1/4-in., 2x50	15-JAN-10	15	8	39.95	0	23119	95	0
1558-QW1	Hrd. cloth, 1/2-in., 3x50	15-JAN-10	23	5	43.99	0	23119	25	0
2232/QTY	B&D jigsaw, 12-in. blade	30-DEC-09	8	5	189.92	.05	24288	15	0
2232/QWE	B&D jigsaw, 8-in. blade	24-DEC-09	6	5	99.87	.05	24288	15	0
2238/QPD	B&D cordless drill, 1/2-in.	20-JAN-10	12	5	38.95	.05	25595	12	0
23109-HB	Claw hammer	20-JAN-10	23	10	9.95	.1	21225	25	0
23114-AA	Sledge hammer, 12 lb.	02-JAN-10	8	5	14.4	.05		12	0
54778-2T	Rat-tail file, 1/8-in. fine	15-DEC-09	43	20	4.99	0	21344	25	0
89-WRE-Q	Hicut chain saw, 16 in.	07-FEB-10	11	5	256.99	.05	24288	10	0
PUC230BT	PVC pipe, 3.5-in., 8-Ft	20-FEB-10	180	75	5.87	0		50	0
SM-18277	1.25-in. metal screw, 25	01-MAR-10	172	75	6.99	0	21225	50	0
SW-23116	2.5-in. wd. screw, 50	24-FEB-10	237	100	8.45	0	21231	100	0
WR3/TI3	Steel matting, 4'x8'x1/6", .5"	17-JAN-10	18	5	119.95	.1	25595	10	0

16 rows selected.

- ▶ A product's quantity on hand (P_QOH) is updated when the product is sold, the system should automatically check whether the quantity on hand falls below its minimum allowable quantity (P_MIN).

Triggers

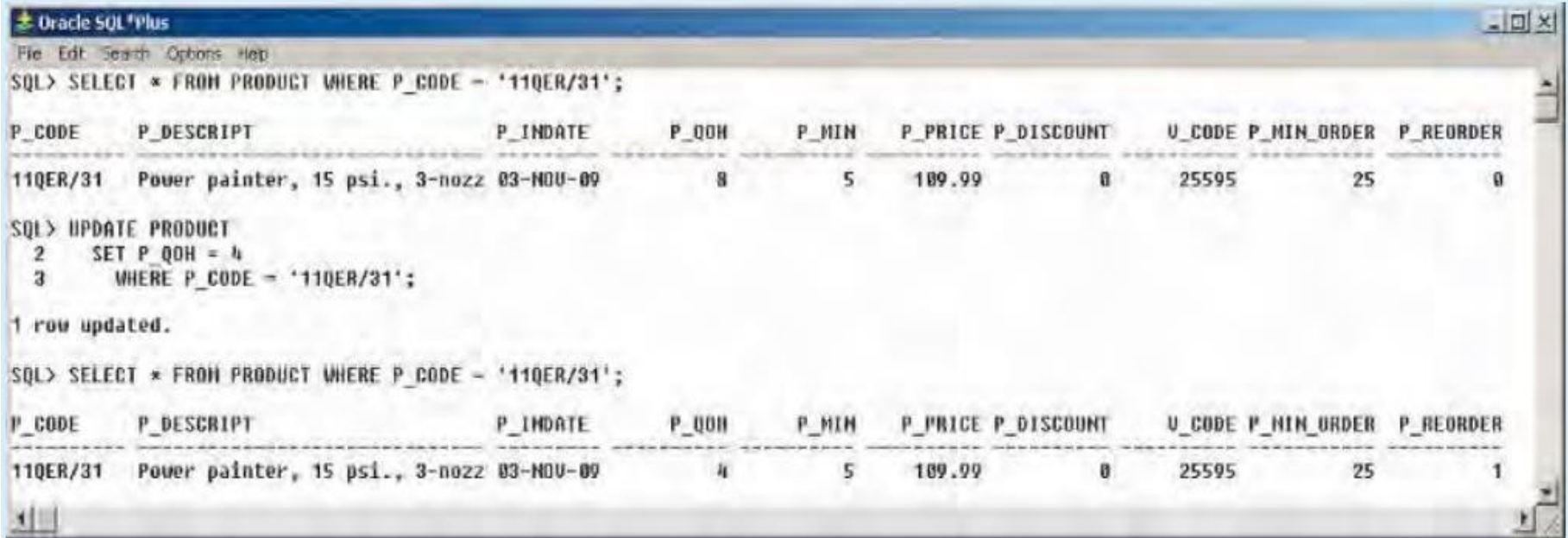
Creating the TRG_PRODUCT_REORDER trigger



```
Oracle SQL*Plus
File Edit Search Options Help
SQL> CREATE OR REPLACE TRIGGER TRG_PRODUCT_REORDER
  2  AFTER INSERT OR UPDATE OF P_QOH ON PRODUCT
  3  BEGIN
  4      UPDATE PRODUCT
  5          SET P_REORDER = 1
  6          WHERE P_QOH <= P_MIN;
  7  END;
  8  /

Trigger created.
```

After the UPDATE completes, the trigger is automatically fired and the UPDATE statement (inside the trigger code) sets the P_REORDER to 1 for all products that are below the minimum.



```
Oracle SQL*Plus
File Edit Search Options Help
SQL> SELECT * FROM PRODUCT WHERE P_CODE = '11QER/31';

P_CODE      P_DESCRIPT      P_INDATE      P_QOH      P_MIN      P_PRICE P_DISCOUNT      U_CODE P_MIN_ORDER  P_REORDER
-----
11QER/31    Power painter, 15 psi., 3-nozz 03-NOV-09           8           5      109.99           0      25595          25           0

SQL> UPDATE PRODUCT
2   SET P_QOH = 4
3   WHERE P_CODE = '11QER/31';

1 row updated.

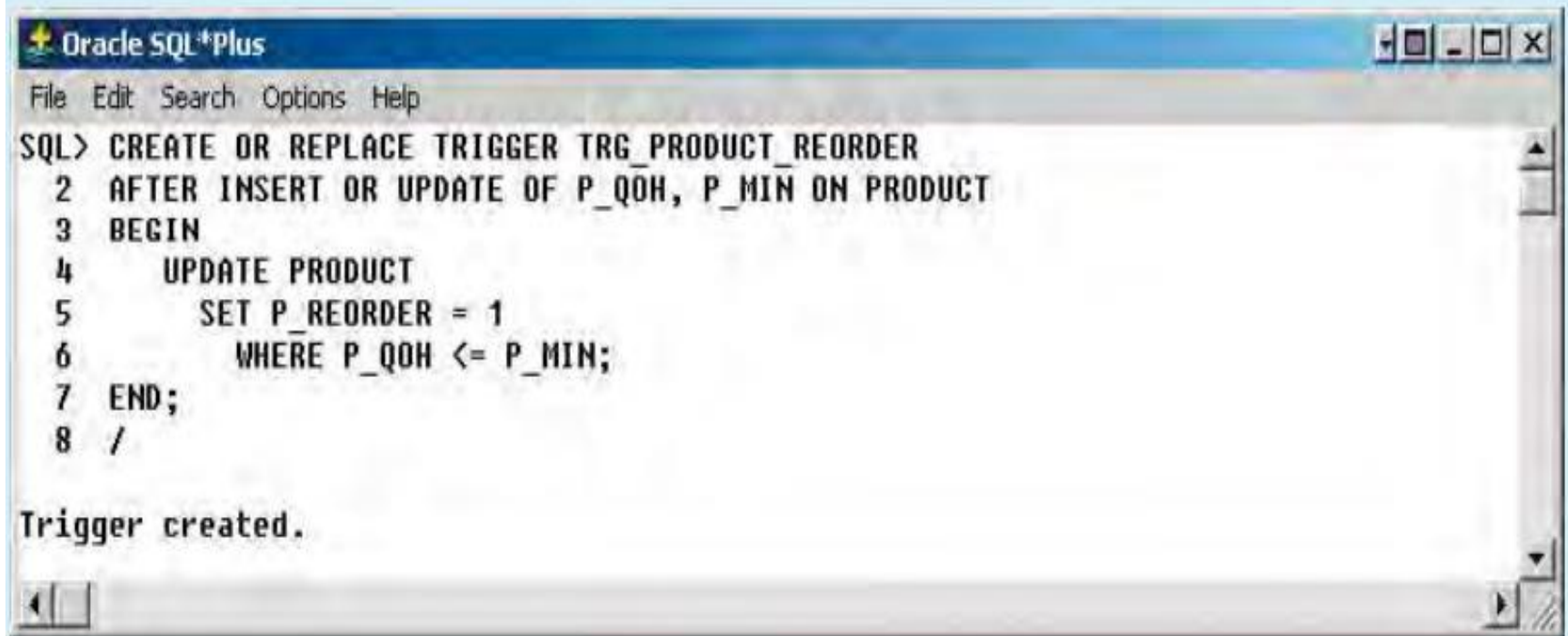
SQL> SELECT * FROM PRODUCT WHERE P_CODE = '11QER/31';

P_CODE      P_DESCRIPT      P_INDATE      P_QOH      P_MIN      P_PRICE P_DISCOUNT      U_CODE P_MIN_ORDER  P_REORDER
-----
11QER/31    Power painter, 15 psi., 3-nozz 03-NOV-09           4           5      109.99           0      25595          25           1
```

- ▶ To test the TRG_PRODUCT_REORDER trigger, let's update the quantity on hand of product '11QER/31' to 4.
- ▶ After the UPDATE completes, the trigger is automatically fired and the UPDATE statement (inside the trigger code) sets the P_REORDER to 1 for all products that are below the minimum.

Triggers

Second version of the TRG_PRODUCT_REORDER trigger

A screenshot of an Oracle SQL*Plus window. The title bar reads "Oracle SQL*Plus". The menu bar includes "File", "Edit", "Search", "Options", and "Help". The main text area contains the following SQL code:

```
SQL> CREATE OR REPLACE TRIGGER TRG_PRODUCT_REORDER
  2  AFTER INSERT OR UPDATE OF P_QOH, P_MIN ON PRODUCT
  3  BEGIN
  4      UPDATE PRODUCT
  5          SET P_REORDER = 1
  6          WHERE P_QOH <= P_MIN;
  7  END;
  8  /
```

Below the code, the message "Trigger created." is displayed. The window has standard Windows-style window controls (minimize, maximize, close) in the top right corner and a status bar at the bottom.


```

Oracle SQL*Plus
File Edit Search Options Help
SQL> SELECT * FROM PRODUCT WHERE P_CODE = '23114-AA';

P_CODE      P_DESCRIPT      P_INDATE      P_QOH      P_MIN      P_PRICE P_DISCOUNT      U_CODE P_MIN_ORDER  P_REORDER
-----
23114-AA    Sledge hammer, 12 lb.    02-JAN-10      8          5          14.4      .05              12      0

SQL> UPDATE PRODUCT
2   SET P_MIN = 10
3   WHERE P_CODE = '23114-AA';

1 row updated.

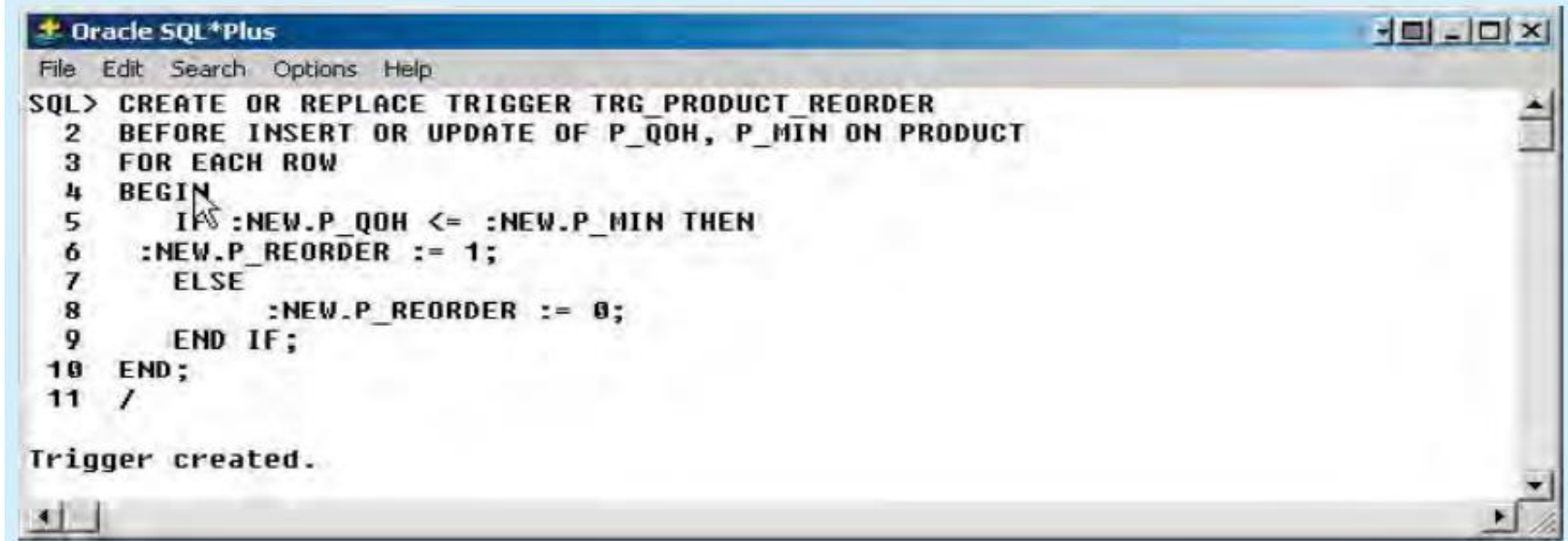
SQL> SELECT * FROM PRODUCT WHERE P_CODE = '23114-AA';

P_CODE      P_DESCRIPT      P_INDATE      P_QOH      P_MIN      P_PRICE P_DISCOUNT      U_CODE P_MIN_ORDER  P_REORDER
-----
23114-AA    Sledge hammer, 12 lb.    02-JAN-10      8          10          14.4      .05              12      1

```

- ▶ To test this new trigger version, let's change the minimum quantity for product '23114-AA' to 10. After that update, the
- ▶ Trigger makes sure that the reorder flag is properly set for all of the products in the PRODUCT table.

The third version of the TRG_PRODUCT_REORDER trigger

A screenshot of an Oracle SQL*Plus window. The title bar reads "Oracle SQL*Plus". The menu bar includes "File", "Edit", "Search", "Options", and "Help". The main text area contains the following SQL code:

```
SQL> CREATE OR REPLACE TRIGGER TRG_PRODUCT_REORDER
2  BEFORE INSERT OR UPDATE OF P_QOH, P_MIN ON PRODUCT
3  FOR EACH ROW
4  BEGIN
5      IF :NEW.P_QOH <= :NEW.P_MIN THEN
6          :NEW.P_REORDER := 1;
7      ELSE
8          :NEW.P_REORDER := 0;
9      END IF;
10 END;
11 /
```

Below the code, the message "Trigger created." is displayed. The window has standard Windows-style window controls (minimize, maximize, close) in the top right corner and a status bar at the bottom.

- ▶ The trigger is executed before the actual triggering statement is completed.
- ▶ The trigger is a row-level trigger instead of a statement-level trigger. Therefore, this trigger executes once for each row affected by the triggering statement.
- ▶ The trigger action uses the :NEW attribute reference to change the value of the P_REORDER attribute.

**FIGURE
8.38**

Execution of the third trigger version

Oracle SQL*Plus

File Edit Search Options Help

SQL> SELECT * FROM PRODUCT;

P_CODE	P_DESCRIPTION	P_INDATE	P_QOH	P_MIN	P_PRICE	P_DISCOUNT	U_CODE	P_MIN_ORDER	P_REORDER
11QER/31	Power painter, 15 psi., 3-nozz	03-NOV-09	29	5	109.99	.0	25595	25	1
13-Q2/P2	7.25-in. pwr. saw blade	13-DEC-09	32	15	14.99	.05	21344	50	0
14-Q1/L3	9.00-in. pwr. saw blade	13-NOV-09	18	12	17.49	.0	21344	50	0
1546-QQ2	Hrd. cloth, 1/4-in., 2x50	15-JAN-10	15	8	39.95	.0	23119	35	0
1558-QW1	Hrd. cloth, 1/2-in., 3x50	15-JAN-10	23	5	43.99	.0	23119	25	0
2232/QTY	B&D jigsaw, 12-in. blade	30-DEC-09	8	5	109.92	.05	24288	15	0
2232/QWE	B&D jigsaw, 8-in. blade	24-DEC-09	6	7	99.87	.05	24288	15	1
2238/QPD	B&D cordless drill, 1/2-in.	20-JAN-10	12	5	38.95	.05	25595	12	0
23109-HB	Claw hammer	20-JAN-10	29	10	9.95	.1	21225	25	0
23114-AA	Sledge hammer, 12 lb.	02-JAN-10	8	10	14.4	.05		12	1
54778-2T	Rat-tail file, 1/8-in. fine	15-DEC-09	43	20	4.99	.0	21344	25	0
89-WRE-Q	Hicut chain saw, 16 in.	07-FEB-10	11	5	256.99	.05	24288	10	0
PVC23DRT	PVC pipe, 3.5-in., 8-ft	20-FEB-10	188	75	5.87	.0		50	0
SH-18277	1.25-in. metal screw, 25	01-MAR-10	172	75	6.99	.0	21225	50	0
SW-23116	2.5-in. wd. screw, 50	24-FEB-10	237	100	8.45	.0	21231	100	0
WR3/TT3	Steel matting, 4'x8'x1/6", .5"	17-JAN-10	10	5	119.95	.1	25595	10	0

16 rows selected.

SQL> UPDATE PRODUCT SET P_QOH = P_QOH;

16 rows updated.

SQL> SELECT * FROM PRODUCT WHERE P_CODE = '11QER/31';

P_CODE	P_DESCRIPTION	P_INDATE	P_QOH	P_MIN	P_PRICE	P_DISCOUNT	U_CODE	P_MIN_ORDER	P_REORDER
11QER/31	Power painter, 15 psi., 3-nozz	03-NOV-09	29	5	109.99	.0	25595	25	0

Stored Procedures

- ▶ Named collection of procedural and SQL statements.
- ▶ Stored procedures are stored in the database.
- ▶ One of the major advantages of stored procedures is that they can be used to encapsulate and represent business transactions.
- ▶ Advantages
 - ▶ Substantially reduce network traffic and increase performance
 - ▶ No transmission of individual SQL statements over network
 - ▶ Reduce code duplication by means of code isolation and code sharing
 - ▶ Minimize chance of errors and cost of application development and maintenance

Stored Procedures

- ▶ Syntax to create a stored procedure:

```
CREATE OR REPLACE PROCEDURE procedure_name [(argument  
    [IN/OUT] data-type, )]  
    [IS/AS]  
    [variable_name data type[:=initial_value] ]  
  
BEGIN  
    PL/SQL or SQL statements;  
    ...  
END;
```

Stored Procedures

- ▶ Points about stored procedures and their syntax:
 - ▶ ***argument** specifies the parameters that are passed to the stored procedure. A stored procedure could have zero or more arguments or parameters.*
 - ▶ ***IN/OUT** indicates whether the parameter is for input, output, or both.*
 - ▶ ***data-type** is one of the procedural SQL data types used in the RDBMS.*
 - ▶ ***Variables** can be declared between the keywords IS and BEGIN. You must specify the variable name, its data type, and (optionally) an initial value.*

Stored Procedures

Creating the PRC_PROD_DISCOUNT stored procedure



```
Oracle SQL*Plus
File Edit Search Options Help
SQL> CREATE OR REPLACE PROCEDURE PRC_PROD_DISCOUNT
2  AS BEGIN
3      UPDATE PRODUCT
4          SET P_DISCOUNT = P_DISCOUNT + .05
5          WHERE P_QOH >= P_MIN*2;
6      DBMS_OUTPUT.PUT_LINE ('* * Update finished * *');
7  END;
8  /

Procedure created.
```

- It assign an additional 5 percent discount for all products when the quantity on hand is more than or equal to twice the minimum quantity.
- To execute the stored procedure, you must use the following syntax: `EXEC procedure_name[(parameter_list)];`

Oracle SQL*Plus									
File Edit Search Options Help									
SQL> SELECT * FROM PRODUCT;									
P_CODE	P_DESCRIPT	P_INDATE	P_QOH	P_MIN	P_PRICE	P_DISCOUNT	U_CODE	P_MIN_ORDER	P_REORDER
11VER/31	Power painter, 15 psi., 3-nozz	03-NOV-09	29	5	109.99	.00	25595	25	0
13-Q2/P2	7.25-in. pow. saw blade	13-DEC-09	32	15	14.99	.05	21344	50	0
14-Q1/L3	9.00-in. pow. saw blade	13-NOV-09	18	12	17.49	.00	21344	50	0
1546-QQ2	Hrd. cloth, 1/4-in., 2x50	15-JAN-10	15	0	39.95	.00	23119	35	0
1558-QW1	Hrd. cloth, 1/2-in., 3x50	15-JAN-10	23	5	43.99	.00	23119	25	0
2232/QTV	B&D jigsaw, 12-in. blade	30-DEC-09	8	5	109.92	.05	24280	15	0
2232/QWC	B&D jigsaw, 8-in. blade	24-DEC-09	6	7	99.87	.05	24280	15	1
2238/QPD	B&D cordless drill, 1/2-in.	20-JAN-10	12	5	38.95	.05	25595	12	0
23109-HB	Claw hammer	20-JAN-10	23	10	9.95	.10	21225	25	0
23114-AA	Sledge hammer, 12 lb.	02-JAN-10	8	10	14.4	.05		12	1
54778-2T	Rat-tail file, 1/8-in. fine	15-DEC-09	43	20	4.99	.00	21344	25	0
H9-WRE-Q	Hicut chain saw, 16 in.	07-FEB-10	11	5	256.99	.05	24280	10	0
PVC23DRT	PVC pipe, 3.5-in., 8-ft	20-FEB-10	188	75	5.87	.00		50	0
SN-18277	1.25-in. metal screw, 25	01-MAR-10	172	75	6.99	.00	21225	50	0
SW-23116	2.5-in. wd. screw, 50	24-FEB-10	207	100	8.45	.00	21231	100	0
WR3/TT3	Steel matting, 4'x8'x1/8", .5"	17-JAN-10	18	5	119.95	.10	25595	10	0
16 rows selected.									

```
SQL> EXEC PAC_PROD_DISCOUNT;
```

```
** Update finished **
```

```
PL/SQL procedure successfully completed.
```

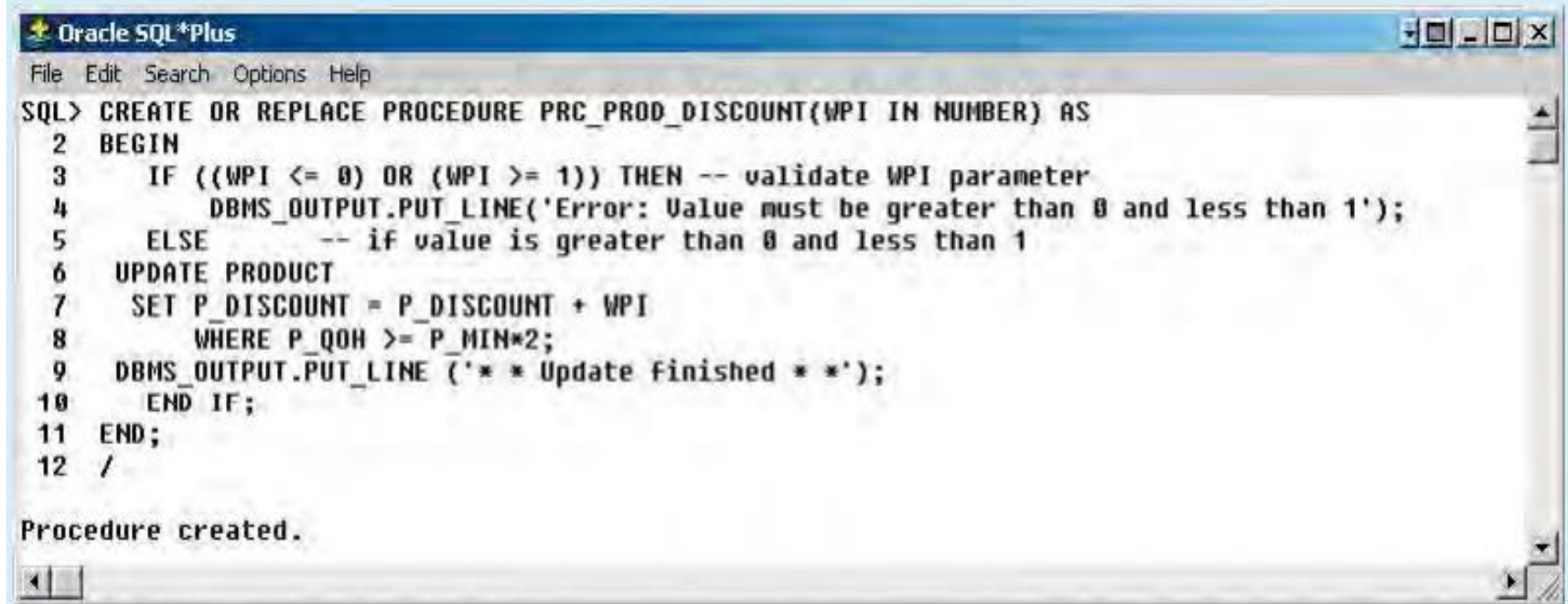
```
SQL> SELECT * FROM PRODUCT;
```

P_CODE	P_DESCRIPT	P_INDATE	P_QUN	P_MIN	P_PRICE	P_DISCOUNT	V_CODE	P_MIN_ORDER	P_REORDER
11QER/31	Power painter, 15 psi., 3-nozz	03-NOV-09	29	5	109.99	.05	25595	25	0
13-Q2/P2	7.25-in. pur. saw blade	13-DEC-09	32	15	14.99	.10	21344	50	0
14-Q1/L3	9.00-in. pur. saw blade	13-NOV-09	18	12	17.49	.00	21344	50	0
1546-QQ2	Hrd. cloth, 1/4-in., 2x50	15-JAN-10	15	8	39.95	.00	23119	35	0
1550-QW1	Hrd. cloth, 1/2-in., 3x50	15-JAN-10	29	5	43.99	.05	23119	25	0
2232/QTY	B&B jigsaw, 12-in. blade	30-DEC-07	8	5	109.92	.05	24288	15	0
2232/QWE	B&B jigsaw, 8-in. blade	24-DEC-09	6	7	99.87	.05	24288	15	1
2238/QPD	B&B cordless drill, 1/2-in.	20-JAN-10	12	5	38.95	.10	25595	12	0
23109-H0	Claw hammer	20-JAN-10	20	10	9.95	.15	21225	25	0
23114-AA	Sledge hammer, 12 lb.	02-JAN-10	8	10	14.4	.05		12	1
54778-21	Rat-tail file, 1/8-in. fine	15-DEC-09	43	20	4.99	.05	21344	25	0
89-WRE-Q	Wicut chain saw, 16 in.	07-FEB-10	11	5	256.99	.10	24288	10	0
PUC23DRT	PUC pipe, 3.5-in., 8-ft	20-FEB-10	188	75	5.87	.05		50	0
SM-18277	1.25-in. metal screw, 25	01-MAR-10	172	75	6.99	.05	21225	50	0
SW-23116	2.5-in. wd. screw, 50	24-FEB-10	237	100	8.45	.05	21231	100	0
WA3/TT3	Steel matting, 4'x8'x1/8", .5"	17-JAN-10	18	5	119.95	.15	25595	10	0

```
16 rows selected.
```


Stored Procedures

RE Second version of the PRC_PROD_DISCOUNT stored procedure

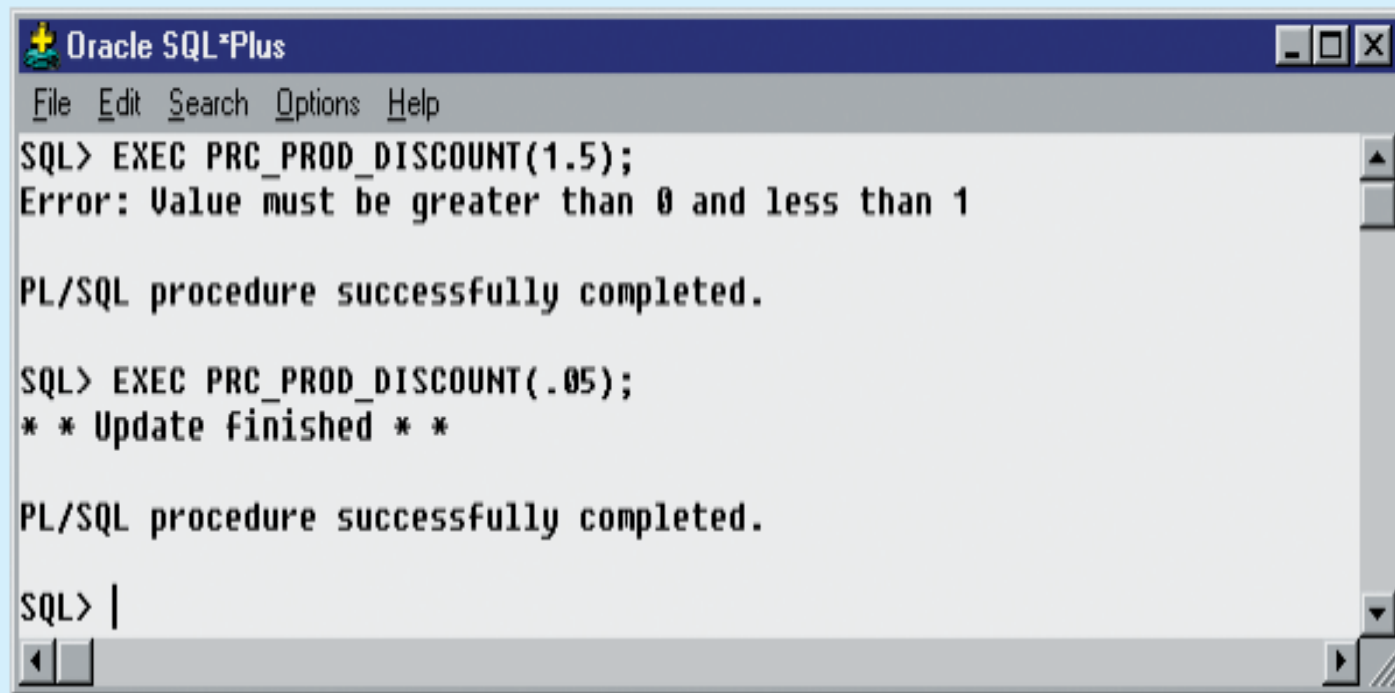


```
Oracle SQL*Plus
File Edit Search Options Help
SQL> CREATE OR REPLACE PROCEDURE PRC_PROD_DISCOUNT(WPI IN NUMBER) AS
2 BEGIN
3   IF ((WPI <= 0) OR (WPI >= 1)) THEN -- validate WPI parameter
4     DBMS_OUTPUT.PUT_LINE('Error: Value must be greater than 0 and less than 1');
5   ELSE -- if value is greater than 0 and less than 1
6     UPDATE PRODUCT
7     SET P_DISCOUNT = P_DISCOUNT + WPI
8     WHERE P_QOH >= P_MIN*2;
9     DBMS_OUTPUT.PUT_LINE ('* * Update finished * *');
10  END IF;
11 END;
12 /

Procedure created.
```

Stored Procedures

Results of the second version of the PRC_PROD_DISCOUNT stored procedure



```
Oracle SQL*Plus
File Edit Search Options Help
SQL> EXEC PRC_PROD_DISCOUNT(1.5);
Error: Value must be greater than 0 and less than 1

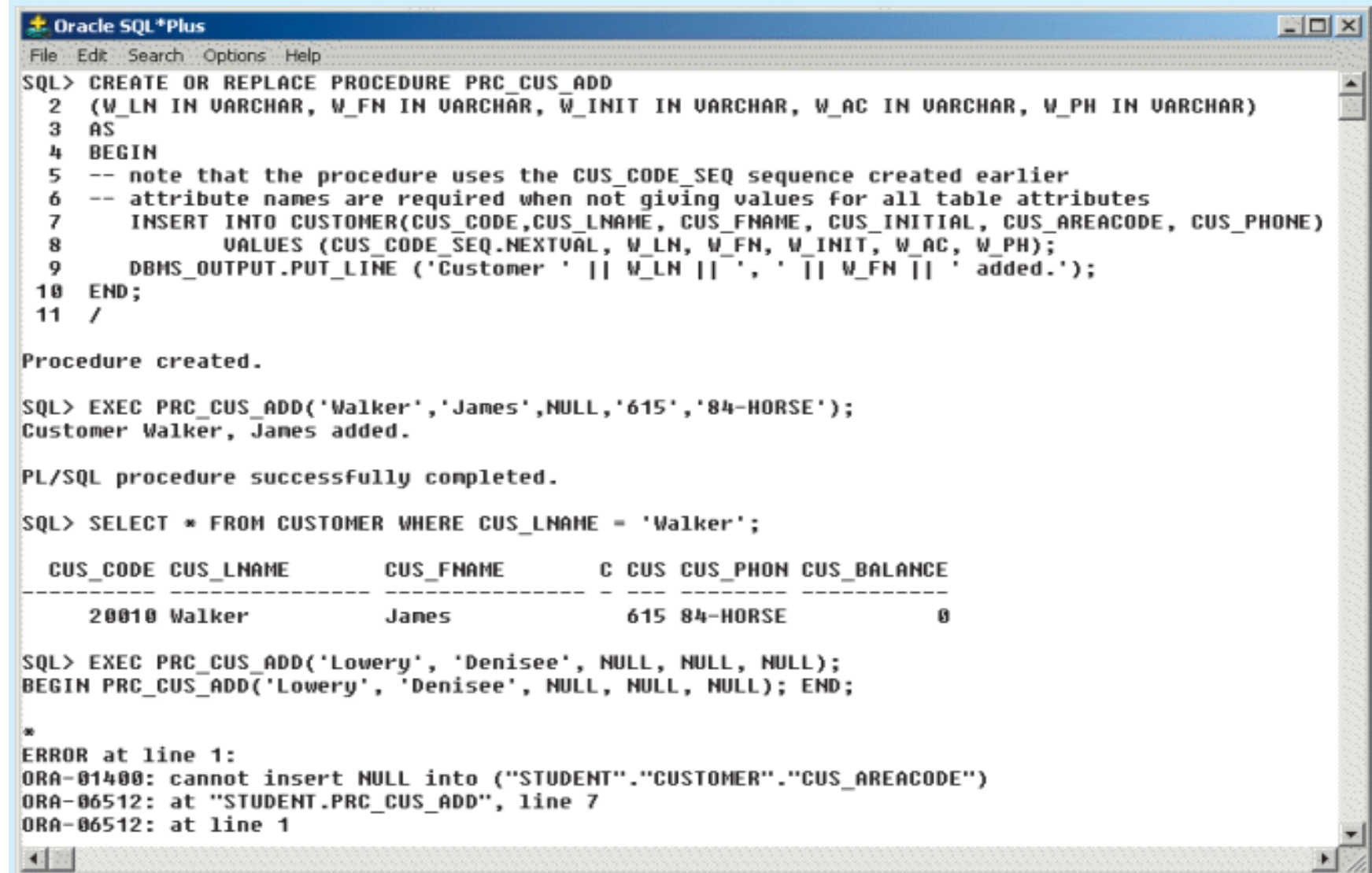
PL/SQL procedure successfully completed.

SQL> EXEC PRC_PROD_DISCOUNT(.05);
* * Update finished * *

PL/SQL procedure successfully completed.

SQL> |
```


FIGURE 5 The PRC_CUS_ADD stored procedure



```
Oracle SQL*Plus
File Edit Search Options Help
SQL> CREATE OR REPLACE PROCEDURE PRC_CUS_ADD
2  (W_LN IN VARCHAR, W_FN IN VARCHAR, W_INIT IN VARCHAR, W_AC IN VARCHAR, W_PH IN VARCHAR)
3  AS
4  BEGIN
5  -- note that the procedure uses the CUS_CODE_SEQ sequence created earlier
6  -- attribute names are required when not giving values for all table attributes
7  INSERT INTO CUSTOMER(CUS_CODE,CUS_LNAME, CUS_FNAME, CUS_INITIAL, CUS_AREACODE, CUS_PHONE)
8  VALUES (CUS_CODE_SEQ.NEXTVAL, W_LN, W_FN, W_INIT, W_AC, W_PH);
9  DBMS_OUTPUT.PUT_LINE ('Customer ' || W_LN || ', ' || W_FN || ' added.');
```

10 END;

11 /

Procedure created.

SQL> EXEC PRC_CUS_ADD('Walker','James',NULL,'615','84-HORSE');

Customer Walker, James added.

PL/SQL procedure successfully completed.

SQL> SELECT * FROM CUSTOMER WHERE CUS_LNAME = 'Walker';

CUS_CODE	CUS_LNAME	CUS_FNAME	C	CUS	CUS_PHON	CUS_BALANCE
20010	Walker	James		615	84-HORSE	0

SQL> EXEC PRC_CUS_ADD('Lowery', 'Denisee', NULL, NULL, NULL);

BEGIN PRC_CUS_ADD('Lowery', 'Denisee', NULL, NULL, NULL); END;

*

ERROR at line 1:

ORA-01400: cannot insert NULL into ("STUDENT"."CUSTOMER"."CUS_AREACODE")

ORA-06512: at "STUDENT.PRC_CUS_ADD", line 7

ORA-06512: at line 1

PL/SQL Processing with Cursors

- ▶ Trigger or stored procedure have returned a single value. If the SQL statement returns more than one value, an error will generate.
- ▶ If you want to use an SQL statement that returns more than one value inside your PL/SQL code, you need to use a cursor.
- ▶ A **cursor** is a special construct used in procedural SQL to hold the data rows returned by an SQL query.
- ▶ A cursor as a reserved area of memory in which the output of the query is stored, like an array holding columns and rows.
- ▶ **Implicit cursor**: automatically created when SQL returns only one value
- ▶ **Explicit cursor**: holds the output of an SQL statement that may return two or more rows
- ▶ To create an explicit cursor, use the following syntax inside a PL/SQL DECLARE section:
CURSOR cursor_name IS select-query;

PL/SQL Processing with Cursors

TABLE
8.9

Cursor Processing Commands

CURSOR COMMAND	EXPLANATION
OPEN	<p>Opening the cursor executes the SQL command and populates the cursor with data, opening the cursor for processing. The cursor declaration command only reserves a named memory area for the cursor; it doesn't populate the cursor with the data. Before you can use a cursor, you need to open it. For example:</p> <pre>OPEN <i>cursor_name</i></pre>
FETCH	<p>Once the cursor is opened, you can use the FETCH command to retrieve data from the cursor and copy it to the PL/SQL variables for processing. The syntax is:</p> <pre>FETCH <i>cursor_name</i> INTO <i>variable1</i> [, <i>variable2</i>, ...]</pre> <p>The PL/SQL variables used to hold the data must be declared in the DECLARE section and must have data types compatible with the columns retrieved by the SQL command. If the cursor's SQL statement returns five columns, there must be five PL/SQL variables to receive the data from the cursor.</p> <p>This type of processing resembles the one-record-at-a-time processing used in previous database models. The first time you fetch a row from the cursor, the first row of data from the cursor is copied to the PL/SQL variables; the second time you fetch a row from the cursor, the second row of data is placed in the PL/SQL variables; and so on.</p>
CLOSE	<p>The CLOSE command closes the cursor for processing.</p>

PL/SQL Processing with Cursors

Cursor Processing Commands:

▶ **OPEN:**

- ▶ Opening the cursor executes the SQL command and populates the cursor with data, opening the cursor for processing.
- ▶ Before you can use a cursor, you need to open it.
- ▶ For example: `OPEN cursor_name`

▶ **FETCH:**

- ▶ Once the cursor is opened, you can use the `FETCH` command to retrieve data from the cursor and copy it to the PL/SQL variables for processing.
- ▶ The syn tax: `FETCH cursor_name INTO variable1 [, variable2, ...]`
- ▶ The first time you fetch a row from the cursor, the first row of data from the cursor is copied to the PL/SQL variables; the second time you fetch a row from the cursor, the second row of data is placed in the PL/SQL variables; and so on.

▶ **CLOSE:**

- ▶ The `CLOSE` command closes the cursor for processing.

PL/SQL Processing with Cursors

TABLE
8.10

Cursor Attributes

ATTRIBUTE	DESCRIPTION
%ROWCOUNT	Returns the number of rows fetched so far. If the cursor is not OPEN, it returns an error. If no FETCH has been done but the cursor is OPEN, it returns 0.
%FOUND	Returns TRUE if the last FETCH returned a row and FALSE if not. If the cursor is not OPEN, it returns an error. If no FETCH has been done, it contains NULL.
%NOTFOUND	Returns TRUE if the last FETCH did not return any row and FALSE if it did. If the cursor is not OPEN, it returns an error. If no FETCH has been done, it contains NULL.
%ISOPEN	Returns TRUE if the cursor is open (ready for processing) or FALSE if the cursor is closed. Remember, before you can use a cursor, you must open it.

PL/SQL Processing with Cursors

Cursor Attributes:

- ▶ **%ROWCOUNT:**
 - ▶ Returns the number of rows fetched so far.
 - ▶ If the cursor is not OPEN, it returns an error.
 - ▶ If no FETCH has been done but the cursor is OPEN, it returns 0.
- ▶ **%FOUND:**
 - ▶ Returns TRUE if the last FETCH returned a row and FALSE if not.
 - ▶ If the cursor is not OPEN, it returns an error.
 - ▶ If no FETCH has been done, it contains NULL.
- ▶ **%NOTFOUND:**
 - ▶ Returns TRUE if the last FETCH did not return any row and FALSE if it did.
 - ▶ If the cursor is not OPEN, it returns an error.
 - ▶ If no FETCH has been done, it contains NULL.
- ▶ **%ISOPEN:**
 - ▶ Returns TRUE if the cursor is open (ready for processing) or FALSE if the cursor is closed.
 - ▶ Remember, before you can use a cursor, you must open it.

```
Oracle SQL*Plus
File Edit Search Options Help
SQL> CREATE OR REPLACE PROCEDURE PRC_CURSOR_EXAMPLE IS
  2 W_P_CODE PRODUCT.P_CODE%TYPE;
  3 W_P_DESCRIPT PRODUCT.P_DESCRIPT%TYPE;
  4 W_TOT  NUMBER(3);
  5 CURSOR PROD_CURSOR IS
  6     SELECT P_CODE, P_DESCRIPT
  7     FROM PRODUCT
  8     WHERE P_QOH > (SELECT AVG(P_QOH) FROM PRODUCT);
  9 BEGIN
 10 DBMS_OUTPUT.PUT_LINE('PRODUCTS WITH P_QOH > AVG(P_QOH)');
 11 DBMS_OUTPUT.PUT_LINE('=====');
 12 OPEN PROD_CURSOR;
 13 LOOP
 14     FETCH PROD_CURSOR INTO W_P_CODE, W_P_DESCRIPT;
 15     EXIT WHEN PROD_CURSOR%NOTFOUND;
 16     DBMS_OUTPUT.PUT_LINE(W_P_CODE || ' -> ' || W_P_DESCRIPT );
 17 END LOOP;
 18 DBMS_OUTPUT.PUT_LINE('=====');
 19 DBMS_OUTPUT.PUT_LINE('TOTAL PRODUCT PROCESSED ' || PROD_CURSOR%ROWCOUNT);
 20 DBMS_OUTPUT.PUT_LINE('--- END OF REPORT ---');
 21 CLOSE PROD_CURSOR;
 22 END;
 23 /
```

Procedure created.

```
SQL> EXEC PRC_CURSOR_EXAMPLE;
PRODUCTS WITH P_QOH > AVG(P_QOH)
=====
PVC23DRT -> PVC pipe, 3.5-in., 8-ft
SH-18277 -> 1.25-in. metal screw, 25
SW-23116 -> 2.5-in. wd. screw, 50
=====
TOTAL PRODUCT PROCESSED 3
--- END OF REPORT ---
```

PL/SQL procedure successfully completed.

PL/SQL Processing with Cursors

- ▶ Lines 2 and 3 use the %TYPE data type in the variable definition section.
- ▶ Line 5 declares the PROD_CURSOR cursor.
- ▶ Line 12 opens the PROD_CURSOR cursor and populates it.
- ▶ Line 13 uses the LOOP statement to loop through the data in the cursor, fetching one row at a time.
- ▶ Line 14 uses the FETCH command to retrieve a row from the cursor and place it in the respective PL/SQL variables.
- ▶ Line 15 uses the EXIT command to evaluate when there are no more rows in the cursor (using the %NOTFOUND cursor attribute) and to exit the loop.
- ▶ Line 19 uses the %ROWCOUNT cursor attribute to obtain the total number of rows processed.
- ▶ Line 21 issues the CLOSE PROD_CURSOR command to close the cursor

PL/SQL Stored Functions

- ▶ A **stored function** is basically a named group of procedural and SQL statements that returns a value.

- ▶ Syntax to create a function:

```
CREATE FUNCTION function_name (argument IN data-type,   )  
    RETURN data-type [IS]
```

```
BEGIN
```

```
PL/SQL statements;
```

```
...
```

```
RETURN (value or expression);
```

```
END;
```

- ▶ Stored functions can be invoked only from within stored procedures or triggers and cannot be invoked from SQL statements.

PL/SQL Stored Functions - Example

- ▶ CREATE OR REPLACE FUNCTION get_square(p_number IN NUMBER)
RETURN NUMBER
IS
BEGIN
 RETURN p_number * p_number;
END; /
- ▶ DECLARE
 v_number NUMBER := 10;
 v_square NUMBER;
BEGIN
 v_square := get_square(v_number);
 DBMS_OUTPUT.PUT_LINE('The square of ' || v_number || ' is ' || v_square);
END; /

EMBEDDED SQL

- ▶ **Embedded SQL** is a term used to refer to SQL statements that are contained within an application programming language (**host language**) such as Visual Basic .NET, C#, COBOL, or Java.
- ▶ Key differences between SQL and procedural languages:
 - ▶ *Run-time mismatch*
 - ▶ SQL is a nonprocedural, interpreted language; each instruction is parsed, its syntax is checked, and it is executed one instruction at a time.
 - ▶ All of the processing takes place at the server side.
 - ▶ Host language is generally a binary-executable program (compiled program).
 - ▶ The host program typically runs at the client side in its own memory space (different from the DBMS environment).

EMBEDDED SQL

- ▶ ***Processing mismatch:***

- ▶ Conventional programming languages (COBOL, ADA, FORTRAN, PASCAL, C++, and PL/I) process one data element at a time.
- ▶ The host language typically manipulates data one record at a time.

- ▶ ***Data type mismatch:***

- ▶ SQL provides several data types, but some of those data types might not match data types used in different host languages (for example, the date and varchar2 data types).

EMBEDDED SQL

- ▶ To integrate SQL within several programming languages, embedded SQL framework defines:
 - ▶ Standard syntax to identify embedded SQL code within host language (EXEC SQL/END-EXEC).
 - ▶ Standard syntax to identify host variables. All host variables are preceded by a colon (":").
 - ▶ Communication area exchanges status and error information between SQL and host language. This communications area contains two variables—SQLCODE and SQLSTATE.

EMBEDDED SQL

- ▶ Another way to interface host languages and SQL is through the use of a call level interface (CLI), in which the programmer writes to an application programming interface (API).
- ▶ A common CLI in Windows is provided by the Open Database Connectivity (ODBC) interface.

EMBEDDED SQL

- ▶ The following general steps are standard:
 1. The programmer writes embedded SQL code within the host language instructions. The code follows the standard syntax required for the host language and embedded SQL.
 2. A preprocessor is used to transform the embedded SQL into specialized procedure calls that are DBMS- and language-specific.
 3. The program is compiled using the host language compiler. The compiler creates an object code module for the program containing the DBMS procedure calls.
 4. The object code is linked to the respective library modules and generates the executable program.
 5. The executable is run, and the embedded SQL statement retrieves data from the database.

EMBEDDED SQL

- ▶ To embed SQL into a host language, follow this syntax:

EXEC SQL

SQL statement;

END-EXEC.

- ▶ The preceding syntax will work for SELECT, INSERT, UPDATE, and DELETE statements. For example, the following embedded SQL code will delete employee 109, George Smith, from the EMPLOYEE table:

EXEC SQL

DELETE FROM EMPLOYEE WHERE EMP_NUM = 109;

END-EXEC.

EMBEDDED SQL

- ▶ In embedded SQL, all host variables are preceded by a colon (“:”).
- ▶ The host variables may be used to send data from the host language to the embedded SQL, or they may be used to receive the data from the embedded SQL.
- ▶ For example, to delete an employee whose employee number is represented by the host variable W_EMP_NUM,:

EXEC SQL

DELETE FROM EMPLOYEE WHERE EMP_NUM = :W_EMP_NUM;

END-EXEC

EMBEDDED SQL

- ▶ The embedded SQL standard defines a SQL communication area to hold status and error information.
- ▶ In COBOL, such an area is known as the SQLCA area and is defined in the Data Division as follows:

EXEC SQL

INCLUDE SQLCA

END-EXEC.

- ▶ The SQLCA area contains two variables for status and error reporting. Table 8.11 shows some of the main values returned by the variables and their meaning.

EMBEDDED SQL

TABLE
8.11 SQL Status and Error Reporting Variables

VARIABLE NAME	VALUE	EXPLANATION
SQLCODE		Old-style error reporting supported for backward compatibility only; returns an integer value (positive or negative).
	0	Successful completion of command.
	100	No data; the SQL statement did not return any rows or did not select, update, or delete any rows.
	-999	Any negative value indicates that an error occurred.
SQLSTATE		Added by SQL-92 standard to provide predefined error codes; defined as a character string (5 characters long).
	00000	Successful completion of command.
		Multiple values in the format XYYYY where: XX-> represents the class code. YYY-> represents the subclass code.

EMBEDDED SQL

- ▶ The following embedded SQL code illustrates the use of the SQLCODE within a COBOL program.

```
EXEC SQL
```

```
    SELECT EMP_FNAME, EMP_LNAME INTO  
           :W_EMP_FNAME, :W_EMP_LNAME
```

```
    WHERE EMP_NUM = :W_EMP_NUM;
```

```
END-EXEC.
```

```
IF SQLCODE = 0 THEN
```

```
    PERFORM DATA_ROUTINE
```

```
ELSE
```

```
    PERFORM ERROR_ROUTINE
```

```
END-IF
```

EMBEDDED SQL

- ▶ If COBOL is used, the cursor can be declared either in the Working Storage Section or in the Procedure Division.

EXEC SQL

```
DECLARE PROD_CURSOR FOR  
SELECT  P_CODE, P_DESCRIPT  
FROM    PRODUCT  
WHERE   P_QOH > (SELECT AVG(P_QOH) FROM PRODUCT);
```

END-EXEC.

- ▶ Next, you must open the cursor to make it ready for processing:

EXEC SQL

```
OPEN PROD_CURSOR;
```

END-EXEC

EMBEDDED SQL

- ▶ The SQLCODE must be checked to ensure that the FETCH command completed successfully.

```
EXEC SQL
```

```
    FETCH PROD_CURSOR INTO :W_P_CODE,  
    :W_P_DESCRIPT;
```

```
END-EXEC.
```

```
IF SQLCODE = 0 THEN
```

```
    PERFORM DATA_ROUTINE
```

```
ELSE
```

```
    PERFORM ERROR_ROUTINE
```

```
END-IF.
```

- ▶ When all rows have been processed, you close the cursor as follows:

```
EXEC SQL
```

```
    CLOSE PROD_CURSOR;
```

```
END-EXEC.
```

EMBEDDED SQL

▶ Static SQL

- ▶ Embedded SQL in which programmer uses predefined SQL statements and parameters
 - ▶ End users of programs are limited to actions that were specified in application programs
- ▶ SQL statements will not change while application is running.

```
SELECT P_CODE, P_DESCRIPT, P_QOH, P_PRICE  
FROM PRODUCT  
WHERE P_PRICE > 100;
```

EMBEDDED SQL

► **Dynamic SQL**

- SQL statement is not known in advance, but instead is generated at run time
- Program can generate SQL statements at run time that are required to respond to ad hoc queries
- Attribute list and condition are not known until end user specifies them
- Tends to be much slower than static SQL
- Requires more computer resources

```
SELECT :W_ATTRIBUTE_LIST  
FROM :W_TABLE  
WHERE :W_CONDITION;
```


Extensible Markup Language (XML)

Extensible Markup Language (XML)

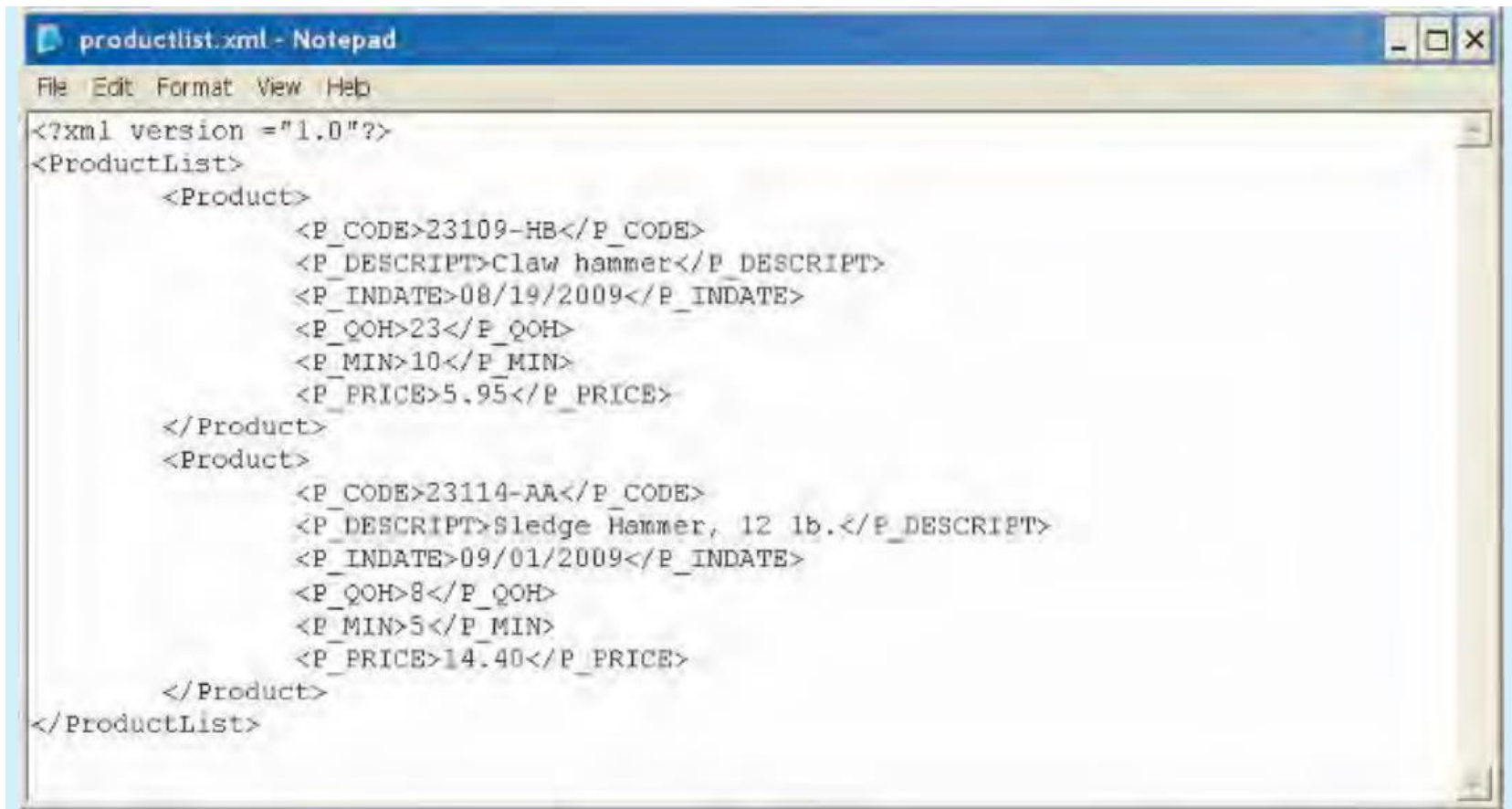
- ▶ Extensible Markup Language (XML) is a meta-language used to represent and manipulate data elements.
- ▶ The World Wide Web Consortium (W3C) published the first XML 1.0 standard definition in 1998.
- ▶ Just like HTML, which was also derived from SGML, an XML document is a text file.

Extensible Markup Language (XML)

- ▶ It has a few very important additional characteristics, as follows:
 - ▶ XML allows the definition of new tags to describe data elements, such as `<ProdctId>`.
 - ▶ XML is case sensitive: `<ProdctId>` is not the same as `<Prodctid>`.
 - ▶ XMLs must be well formed; that is, tags must be properly formatted. Most openings also have a corresponding closing.
 - ▶ XMLs must be properly nested.
 - ▶ You can use the `<--` and `-->` symbols to enter comments in the XML document.
 - ▶ The XML and xml prefixes are reserved for XMLs only

Extensible Markup Language (XML)

- ▶ Consider a B2B example in which Company A uses XML to exchange product data with Company B over the Internet.



```
<?xml version="1.0"?>
<ProductList>
  <Product>
    <P_CODE>23109-HB</P_CODE>
    <P_DESCRIPT>Claw hammer</P_DESCRIPT>
    <P_INDATE>08/19/2009</P_INDATE>
    <P_QOH>23</P_QOH>
    <P_MIN>10</P_MIN>
    <P_PRICE>5.95</P_PRICE>
  </Product>
  <Product>
    <P_CODE>23114-AA</P_CODE>
    <P_DESCRIPT>Sledge Hammer, 12 lb.</P_DESCRIPT>
    <P_INDATE>09/01/2009</P_INDATE>
    <P_QOH>8</P_QOH>
    <P_MIN>5</P_MIN>
    <P_PRICE>14.40</P_PRICE>
  </Product>
</ProductList>
```

Extensible Markup Language (XML)

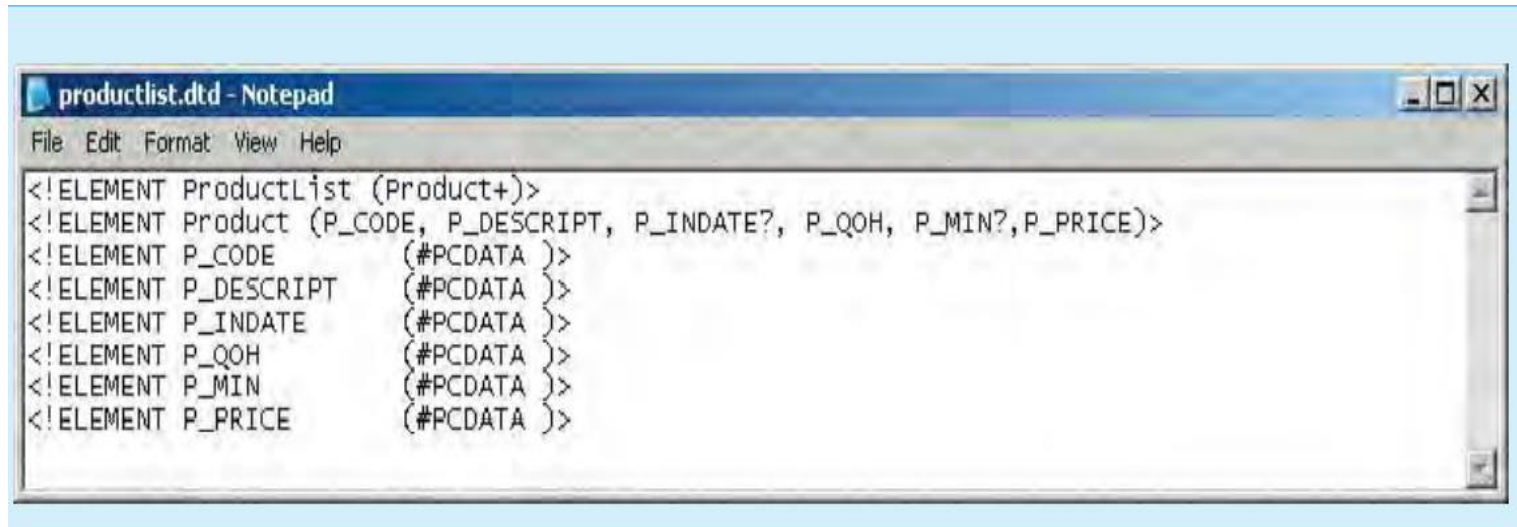
- ▶ The XML example illustrates several important XML features, as follows:
 - ▶ The first line represents the XML document declaration, and it is mandatory.
 - ▶ Every XML document has a root element. In the example, the second line declares the ProductList root element.
 - ▶ The root element contains child elements or subelements. In the example, line 3 declares Product as a child element of ProductList.
 - ▶ Each element can contain subelements. For example, each Product element is composed of several child elements, represented by P_CODE, P_DESCRIPT, P_INDATE, P_QOH, P_MIN, and P_PRICE.
 - ▶ The XML document reflects a hierarchical tree structure where elements are related in a parent-child relationship; each parent element can have many child elements. For example, the root element is ProductList. Product is the child element of ProductList. Product has six child elements: P_CODE, P_DESCRIPT, P_INDATE, P_QOH, P_MIN, and P_PRICE.

Document Type Definitions (DTD) and XML Schemas

- ▶ Companies that use B2B transactions must have a way to understand and validate each other's tags.
- ▶ One way to accomplish that task is through the use of Document Type Definitions.
- ▶ A **Document Type Definition (DTD)** is a file with a .dtd extension that describes XML elements—in effect, a DTD file provides the composition of the database's logical model and defines the syntax rules or valid elements for each type of XML document.
- ▶ Companies that intend to engage in e-commerce business transactions must develop and share DTDs.

Document Type Definitions (DTD) and XML Schemas

- ▶ productlist.dtd document for the productlist.xml



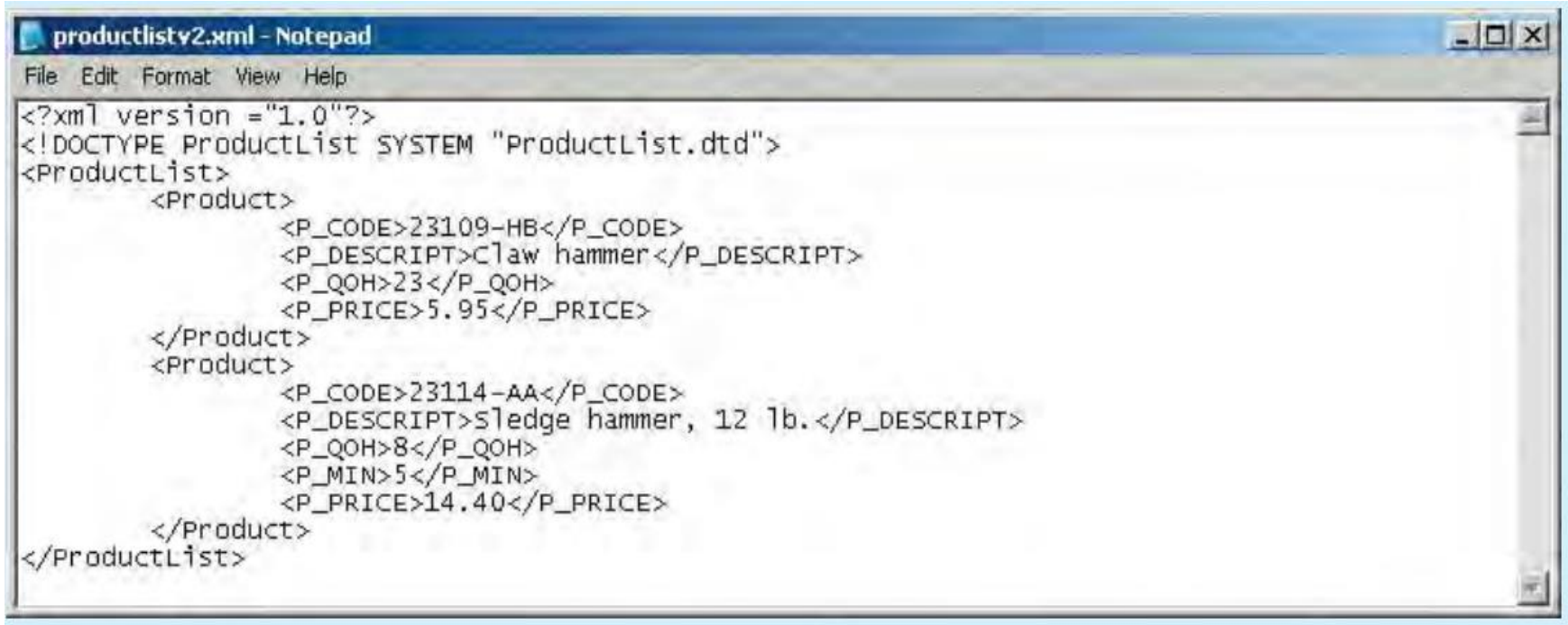
```
productlist.dtd - Notepad
File Edit Format View Help
<!ELEMENT ProductList (Product+)>
<!ELEMENT Product (P_CODE, P_DESCRIPT, P_INDATE?, P_QOH, P_MIN?, P_PRICE)>
<!ELEMENT P_CODE      (#PCDATA )>
<!ELEMENT P_DESCRIPT  (#PCDATA )>
<!ELEMENT P_INDATE    (#PCDATA )>
<!ELEMENT P_QOH       (#PCDATA )>
<!ELEMENT P_MIN       (#PCDATA )>
<!ELEMENT P_PRICE     (#PCDATA )>
```

Document Type Definitions (DTD) and XML Schemas

- ▶ the productlist.dtd file provides definitions of the elements in the productlist.xml document. In particular:
 - ▶ The first line declares the ProductList root element.
 - ▶ The ProductList root element has one child, the Product element.
 - ▶ The plus “+” symbol indicates that Product occurs one or more times within ProductList.
 - ▶ An asterisk “*” would mean that the child element occurs zero or more times.
 - ▶ A question mark “?” would mean that the child element is optional.
 - ▶ The second line describes the Product element.
 - ▶ The question mark “?” after the P_INDATE and P_MIN indicates that they are optional elements.
 - ▶ The third through eighth lines show that the Product element has six child elements.
 - ▶ The #PCDATA keyword represents the actual text data

Document Type Definitions (DTD) and XML Schemas

- ▶ To be able to use a DTD file to define elements within an XML document, the DTD must be referenced from within that XML document.



```
productlistv2.xml - Notepad
File Edit Format View Help
<?xml version="1.0"?>
<!DOCTYPE ProductList SYSTEM "ProductList.dtd">
<ProductList>
  <Product>
    <P_CODE>23109-HB</P_CODE>
    <P_DESCRIPT>Claw hammer</P_DESCRIPT>
    <P_QOH>23</P_QOH>
    <P_PRICE>5.95</P_PRICE>
  </Product>
  <Product>
    <P_CODE>23114-AA</P_CODE>
    <P_DESCRIPT>Sledge hammer, 12 lb.</P_DESCRIPT>
    <P_QOH>8</P_QOH>
    <P_MIN>5</P_MIN>
    <P_PRICE>14.40</P_PRICE>
  </Product>
</ProductList>
```

Document Type Definitions (DTD) and XML Schemas

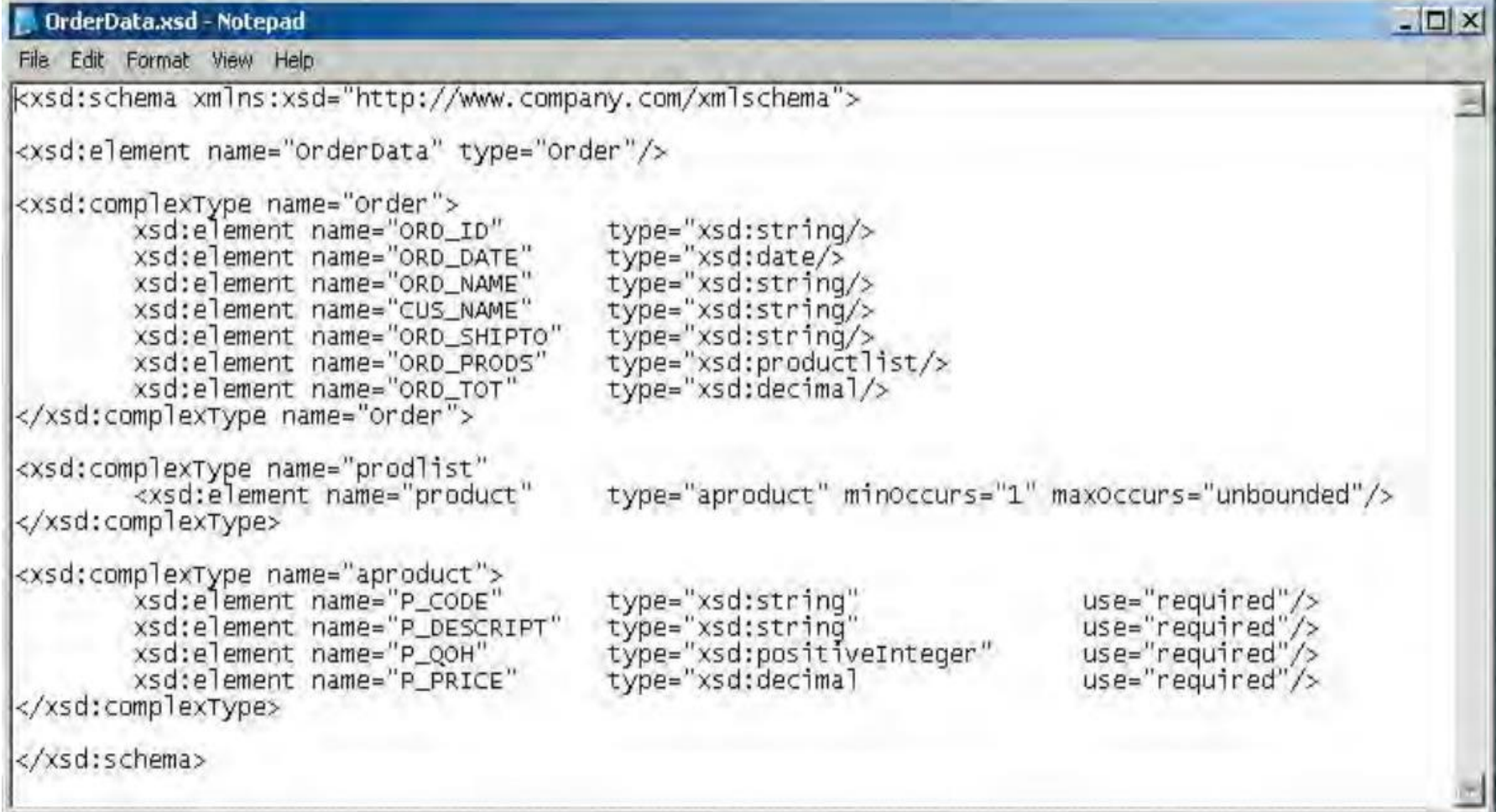
- ▶ P_INDATE and P_MIN do not appear in all Product definitions because they were declared to be optional elements.
- ▶ The DTD can be referenced by many XML documents of the same type. For example, if Company A routinely exchanges product data with Company B, it will need to create the DTD only once.
- ▶ All subsequent XML documents will refer to the DTD, and Company B will be able to verify the data being received.

XML schema

- ▶ The **XML schema** is an advanced data definition language that is used to describe the structure (elements, data types, relationship types, ranges, and default values) of XML data documents.
- ▶ One of the main advantages of an XML schema is that it more closely maps to database terminology and features.
- ▶ For example, an XML schema will be able to define common database types such as date, integer, or decimal; minimum and maximum values; a list of valid values; and required elements.
- ▶ Using the XML schema, a company would be able to validate the data for values that may be out of range, incorrect dates, valid values, and so on.

XML schema

- ▶ Unlike a DTD document, which uses a unique syntax, an XML schema definition (XSD) file uses a syntax that resembles an XML document.
- ▶ Figure 14.14 shows the XSD document for the OrderData XML document.
- ▶ The XML schema syntax is similar to the XML document syntax.
- ▶ In addition, the XML schema introduces additional semantic information for the OrderData XML document, such as string, date, and decimal data types; required elements; and minimum and maximum cardinalities for the data elements.



```
<?xml version='1.0' encoding='UTF-8'>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" >
  <xsd:element name="OrderData" type="order"/>
  <xsd:complexType name="order">
    <xsd:sequence>
      <xsd:element name="ORD_ID" type="xsd:string"/>
      <xsd:element name="ORD_DATE" type="xsd:date"/>
      <xsd:element name="ORD_NAME" type="xsd:string"/>
      <xsd:element name="CUS_NAME" type="xsd:string"/>
      <xsd:element name="ORD_SHIPTO" type="xsd:string"/>
      <xsd:element name="ORD_PRODS" type="xsd:productlist"/>
      <xsd:element name="ORD_TOT" type="xsd:decimal"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="productlist">
    <xsd:sequence>
      <xsd:element name="product" type="aproduct" minOccurs="1" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="aproduct">
    <xsd:sequence>
      <xsd:element name="P_CODE" type="xsd:string" use="required"/>
      <xsd:element name="P_DESCRIPTOR" type="xsd:string" use="required"/>
      <xsd:element name="P_QOH" type="xsd:positiveInteger" use="required"/>
      <xsd:element name="P_PRICE" type="xsd:decimal" use="required"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

XML Presentation

- ▶ One of the main benefits of XML is that it separates data structure from its presentation and processing.
- ▶ By separating data and presentation, you are able to present the same data in different ways—which is similar to having views in SQL.
- ▶ The Extensible Style Language (XSL) specification provides the mechanism to display XML data.
- ▶ XSL is used to define the rules by which XML data are formatted and displayed.
- ▶ The XSL specification is divided in two parts: [Extensible Style Language Transformations \(XSLT\)](#) and [XSL style sheets](#).

XML Presentation

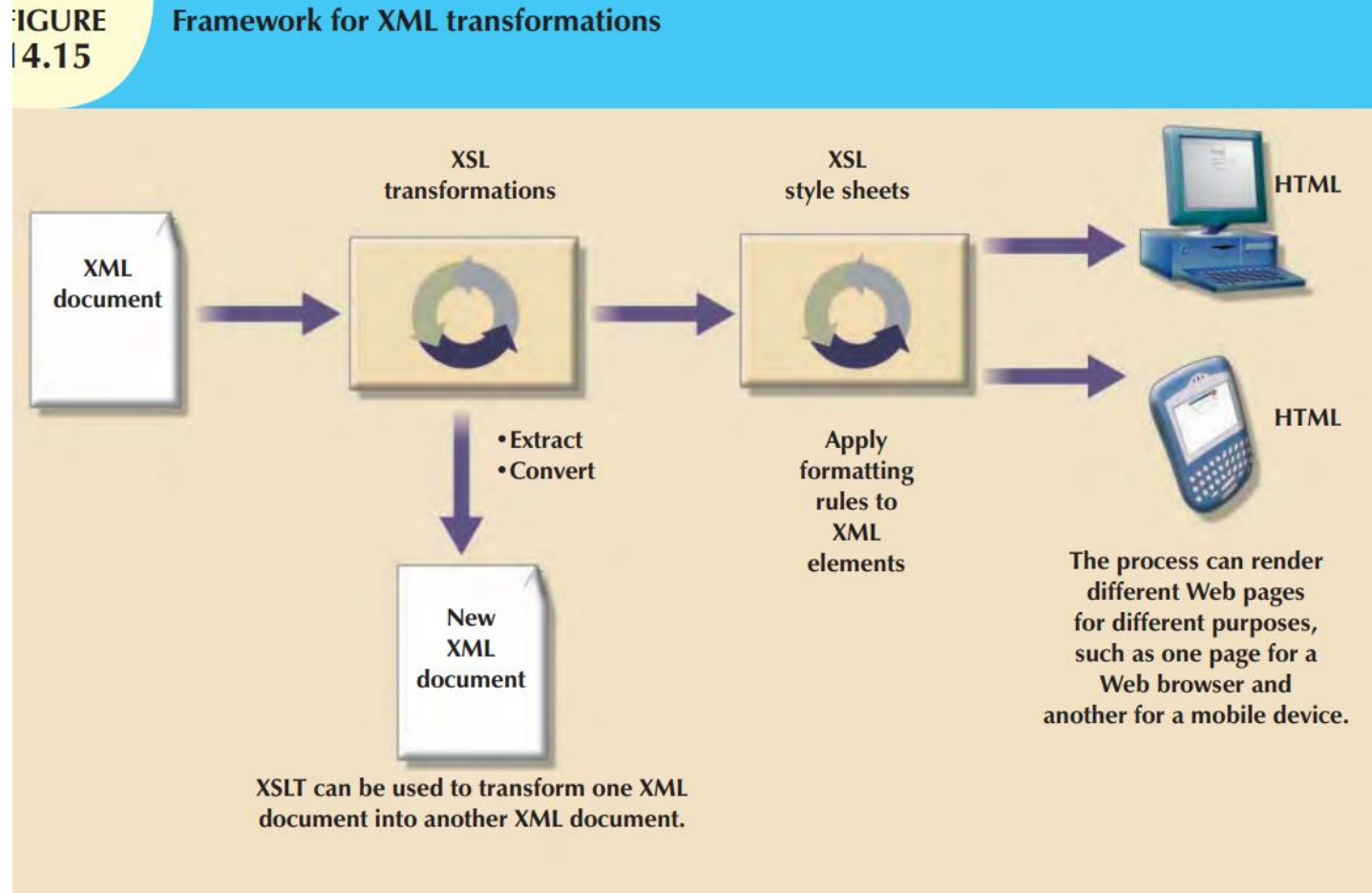
- ▶ **Extensible Style Language Transformations (XSLT)** describe the general mechanism that is used to extract and process data from one XML document and enable its transformation within another document.
- ▶ Using XSLT, you can extract data from an XML document and convert it into a text file, an HTML Web page, or a Web page that is formatted for a mobile device.
- ▶ User sees in those cases is actually a view (or HTML representation) of the actual XML data.
- ▶ XSLT can also be used to extract certain elements from an XML document, such as the product codes and product prices, to create a product catalog.
- ▶ XSLT can even be used to transform one XML document into another XML document.

XML Presentation

- ▶ **XSL style sheets** define the presentation rules applied to XML elements—somewhat like presentation templates.
- ▶ The XSL style sheet describes the formatting options to apply to XML elements when they are displayed on a browser, cellular phone display, PDA screen, and so on.

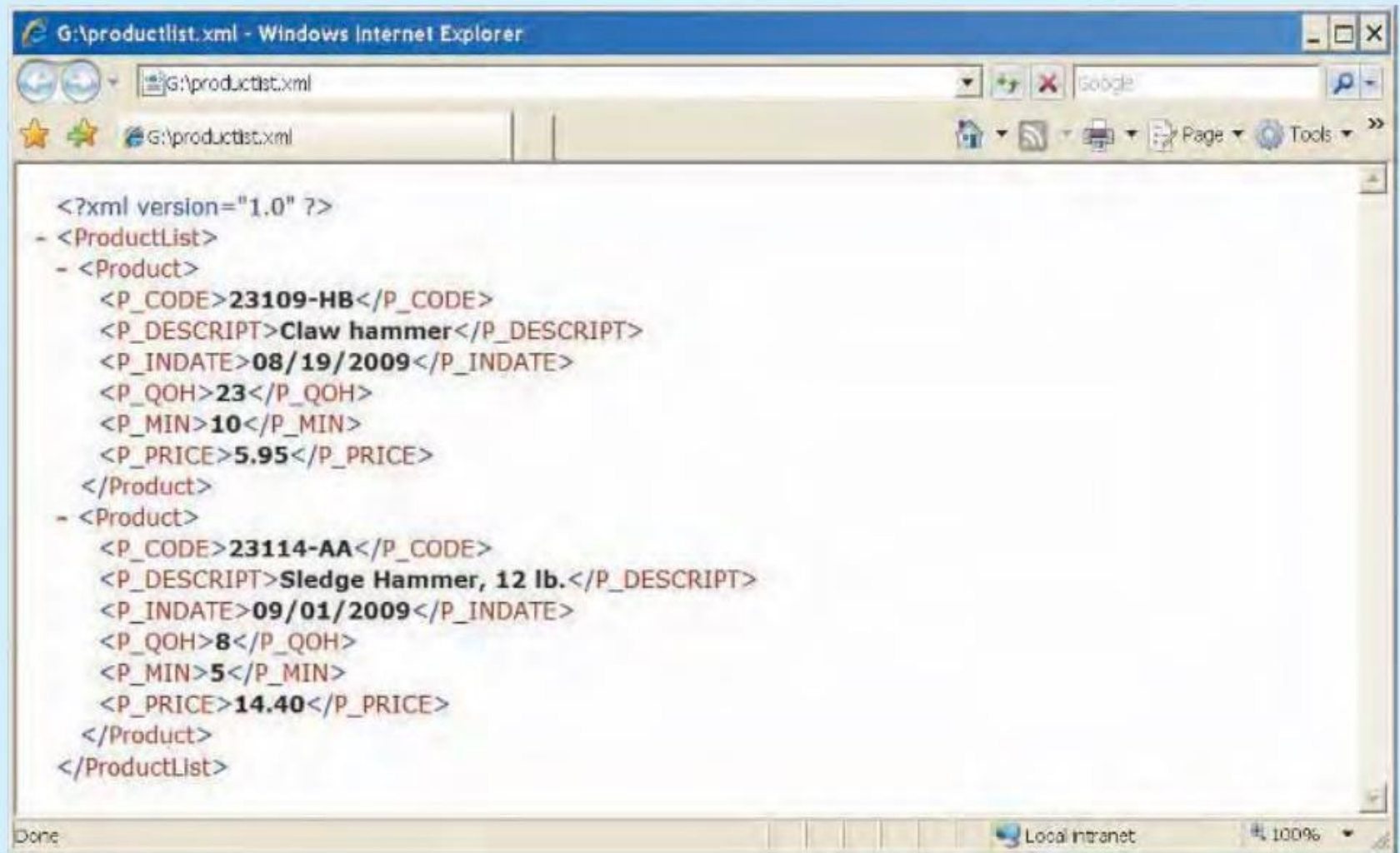
XML Presentation

- ▶ framework used by the various components to translate XML documents into viewable Web pages, an XML document, or some other document.



XML Presentation

- ▶ To display the XML document with Windows Internet Explorer (IE) 5.0 or later, enter the URL of the XML document in the browser's address bar.
- ▶ Figure 14.16 is based on the productlist.xml document created earlier.
- ▶ As you examine Figure 14.16, note that IE shows the XML data in a color-coded, collapsible, treelike structure.



XML Presentation

- ▶ Internet Explorer also provides data binding of XML data to HTML documents.
- ▶ Figure 14.17 shows the HTML code that is used to bind an XML document to an HTML table.

XML data binding

```
productlist.htm - Notepad
File Edit Format View Help

<HTML>
<HEAD>
<TITLE>BINDING THE PRODUCTLIST XML DATA TO HTML TABLE</TITLE>
</HEAD>

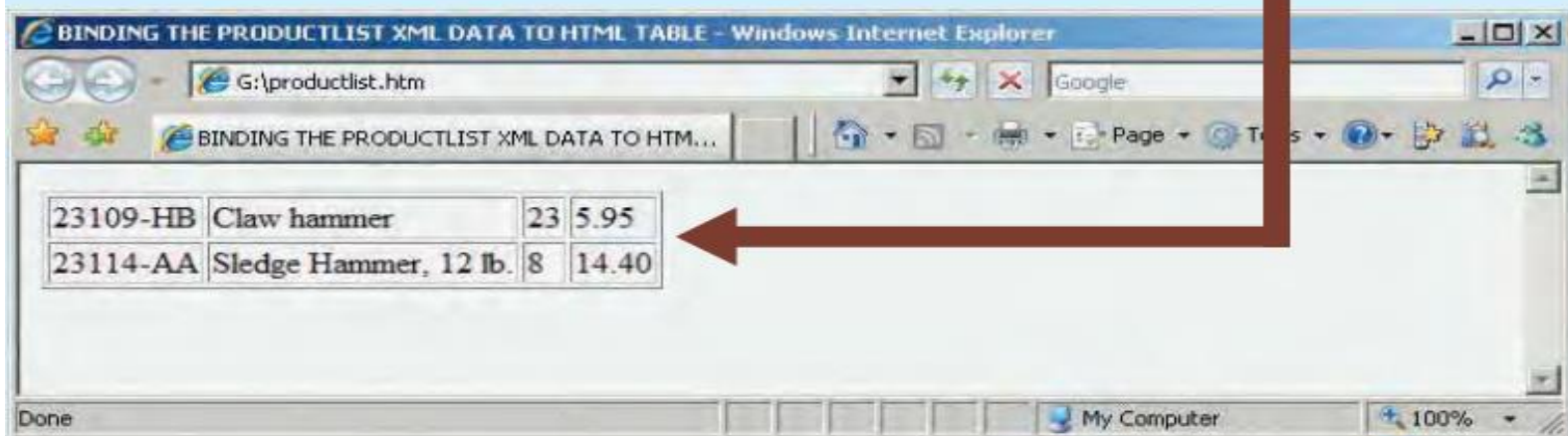
<BODY>

<XML ID="PRODLIST" SRC="PRODUCTLIST.XML"></XML>

<TABLE BORDER="1" DATASRC="#PRODLIST">
  <TR>
    <TD><SPAN DATAFLD="P_CODE"></SPAN></TD>
    <TD><SPAN DATAFLD="P_DESCRIPT"></SPAN></TD>
    <TD><SPAN DATAFLD="P_QOH"></SPAN></TD>
    <TD><SPAN DATAFLD="P_PRICE"></SPAN></TD>
  </TR>

</TABLE>

</BODY>
</HTML>
```



XML Applications

- ▶ **B2B exchanges:** XML enables the exchange of B2B data, providing the standard for all organizations that need to exchange data with partners, competitors, the government, or customers. In particular, XML is positioned to replace EDI as the standard for the automation of the supply chain because **it is less expensive and more flexible**.
- ▶ **Legacy systems integration:** XML provides the “glue” to integrate legacy system data with modern e-commerce Web systems. Web and XML technologies could be used to inject some new life in “old but trusted” legacy applications.
- ▶ **Web page development:** XML provides several features that make it a good fit for certain Web development scenarios. For example, Web portals with large amounts of personalized data can use XML to pull data from multiple external sources (such as news, weather, and stocks) and apply different presentation rules to format pages on desktop computers as well as mobile devices

XML Applications

- ▶ **Database support:** Databases are at the heart of e-commerce applications. A DBMS that supports XML exchanges will be able to integrate with external systems (Web, mobile data, legacy systems, and so on) and thus enable the creation of new types of systems. These databases can import or export data in XML format or generate XML documents from SQL queries while still storing the data, using their native data model format. Alternatively, a DBMS can also support an XML data type to store XML data in its native format.
- ▶ **Database meta-dictionaries:** XML can also be used to create meta-dictionaries, or vocabularies, for databases. These meta-dictionaries can be used by applications that need to access other external data sources. DBMS vendors can publish meta-dictionaries to facilitate data exchanges and the creation of data views from multiple applications—hierarchical, relational, object-oriented, object-relational, or extended relational.

XML Applications

- ▶ **XML databases:** Given the huge number of expected XML-based data exchanges, businesses are already looking for ways to better manage and utilize the data. Currently, many different products are on the market to address this problem. The approaches range from simple middleware XML software, to object databases with XML interfaces, to full XML database engines and servers.. XML databases provide for the storage of data in complex relationships. For example, an XML database would be well suited to store the contents of a book. (a book typically consists of chapters, sections, paragraphs, figures, charts, footnotes, endnotes, and so on.)
- ▶ Examples of XML databases are
 - ▶ Oracle,
 - ▶ IBM DB2,
 - ▶ MS SQL Server,
 - ▶ Ipedo XML Database (www.ipedo.com),
 - ▶ Tamino from Software AG (www.softwareag.com), and
 - ▶ the open source dbXML from <http://sourceforge.net/projects/dbxml-core>.

XML Applications

- ▶ **XML services:** Many companies are already working on the development of a new breed of services based on XML and Web technologies. These services promise to break down the interoperability barriers among systems and companies alike. XML provides the infrastructure that facilitates heterogeneous systems to work together across the desk, the street, and the world. Services would use XML and other Internet technologies to publish their interfaces. Other services, wanting to interact with existing services, would locate them and learn their vocabulary (service request and replies) to establish a “conversation.”

END OF UNIT