



DATABASE SYSTEMS

Design,
Implementation,
and Management
12e

Carlos Coronel | Steven Morris

Chapter 8

Advanced SQL

Learning Objectives

- In this chapter, you will learn:
 - How to use the advanced SQL JOIN operator syntax
 - About the different types of subqueries and correlated queries
 - How to use SQL functions to manipulate dates, strings, and other data
 - About the relational set operators UNION, UNION ALL, INTERSECT, and MINUS

Learning Objectives

- In this chapter, you will learn:
 - How to create and use views and updatable views
 - How to create and use triggers and stored procedures
 - How to create embedded SQL

SQL Join Operators

- Relational join operation merges rows from two tables and returns rows with one of the following:
 - Natural join - common values in common columns
 - Equality or inequality - meet a given join condition
 - Outer join – common values in common columns or no matching values
- **Inner join:** Rows that meet a given criterion are selected
 - Equality condition (natural join or equijoin) or inequality condition (theta join)
- **Outer join:** Returns matching rows and rows with unmatched attribute values for one or both joined tables

Table 8.1 - SQL Join Expression Styles

TABLE 8.1

SQL JOIN EXPRESSION STYLES

JOIN CLASSIFICATION	JOIN TYPE	SQL SYNTAX EXAMPLE	DESCRIPTION
CROSS	CROSS JOIN	SELECT * FROM T1, T2	Returns the Cartesian product of T1 and T2 (old style)
		SELECT * FROM T1 CROSS JOIN T2	Returns the Cartesian product of T1 and T2
INNER	Old-style JOIN	SELECT * FROM T1, T2 WHERE T1.C1=T2.C1	Returns only the rows that meet the join condition in the WHERE clause (old style); only rows with matching values are selected
	NATURAL JOIN	SELECT * FROM T1 NATURAL JOIN T2	Returns only the rows with matching values in the matching columns; the matching columns must have the same names and similar data types
	JOIN USING	SELECT * FROM T1 JOIN T2 USING (C1)	Returns only the rows with matching values in the columns indicated in the USING clause
	JOIN ON	SELECT * FROM T1 JOIN T2 ON T1.C1=T2.C1	Returns only the rows that meet the join condition indicated in the ON clause
OUTER	LEFT JOIN	SELECT * FROM T1 LEFT OUTER JOIN T2 ON T1.C1=T2.C1	Returns rows with matching values and includes all rows from the left table (T1) with unmatched values
	RIGHT JOIN	SELECT * FROM T1 RIGHT OUTER JOIN T2 ON T1.C1=T2.C1	Returns rows with matching values and includes all rows from the right table (T2) with unmatched values
	FULL JOIN	SELECT * FROM T1 FULL OUTER JOIN T2 ON T1.C1=T2.C1	Returns rows with matching values and includes all rows from both tables (T1 and T2) with unmatched values

FIGURE 8.1 NATURAL JOIN RESULTS

The screenshot shows an SQL Plus window with two separate SQL queries and their results.

```
SQL> SELECT CUS_CODE, CUS_LNAME, INV_NUMBER, INV_DATE
  2  FROM CUSTOMER NATURAL JOIN INVOICE;
```

CUS_CODE	CUS_LNAME	INV_NUMBER	INV_DATE
10011	Dunne	1008	17-JAN-16
10011	Dunne	1004	17-JAN-16
10011	Dunne	1002	16-JAN-16
10012	Smith	1003	16-JAN-16
10014	Orlando	1006	17-JAN-16
10014	Orlando	1001	16-JAN-16
10015	O'Brian	1007	17-JAN-16
10018	Farriss	1005	17-JAN-16

8 rows selected.

```
SQL> SELECT INV_NUMBER, P_CODE, P_DESCRPT, LINE_UNITS, LINE_PRICE
  2  FROM INVOICE NATURAL JOIN LINE NATURAL JOIN PRODUCT;
```

INV_NUMBER	P_CODE	P_DESCRPT	LINE_UNITS	LINE_PRICE
1001	13-Q2/P2	7.25-in. pwr. saw blade	1	14.99
1001	23109-HB	Claw hammer	1	9.95
1002	54778-2T	Rat-tail file, 1/8-in. fine	2	4.99
1003	2238/QPD	B&D cordless drill, 1/2-in.	1	38.95
1003	1546-QQ2	Hrd. cloth, 1/4-in., 2x50	1	39.95
1003	13-Q2/P2	7.25-in. pwr. saw blade	5	14.99
1004	54778-2T	Rat-tail file, 1/8-in. fine	3	4.99
1004	23109-HB	Claw hammer	2	9.95
1005	PVC23DRT	PVC pipe, 3.5-in., 8-ft	12	5.87
1006	SM-18277	1.25-in. metal screw, 25	3	6.99
1006	2232/QTY	B&D jigsaw, 12-in. blade	1	109.92
1006	23109-HB	Claw hammer	1	9.95
1006	89-WRE-Q	Hicut chain saw, 16 in.	1	256.99
1007	13-Q2/P2	7.25-in. pwr. saw blade	2	14.99
1007	54778-2T	Rat-tail file, 1/8-in. fine	1	4.99
1008	PVC23DRT	PVC pipe, 3.5-in., 8-ft	5	5.87
1008	WR3/TT3	Steel matting, 4'x8'x1/6", .5" mesh	3	119.95
1008	23109-HB	Claw hammer	1	9.95

18 rows selected.

```
SQL> -
```

FIGURE 8.2 JOIN USING RESULTS

The screenshot shows a SQL query window titled "SQL File 3". The query is:

```
1 • SELECT INV_NUMBER, P_CODE, P_DESCRPT, LINE_UNITS, LINE_PRICE  
2   FROM INVOICE JOIN LINE USING (INV_NUMBER)  
3     JOIN PRODUCT USING (P_CODE);
```

The results grid displays the following data:

	INV_NUMBER	P_CODE	P_DESCRPT	LINE_UNITS	LINE_PRICE
▶	1001	13-Q2/P2	7.25-in. pwr. saw blade	1	14.99
	1001	23109-HB	Claw hammer	1	9.95
	1002	54778-2T	Rat-tail file, 1/8-in. fine	2	4.99
	1003	2238/QPD	B&D cordless drill, 1/2-in.	1	38.95
	1003	1546-QQ2	Hrd. cloth, 1/4-in., 2x50	1	39.95
	1003	13-Q2/P2	7.25-in. pwr. saw blade	5	14.99
	1004	54778-2T	Rat-tail file, 1/8-in. fine	3	4.99
	1004	23109-HB	Claw hammer	2	9.95
	1005	PVC23DRT	PVC pipe, 3.5-in., 8-ft	12	5.87
	1006	SM-18277	1.25-in. metal screw, 25	3	6.99
	1006	2232/QTY	B&D jigsaw, 12-in. blade	1	109.92
	1006	23109-HB	Claw hammer	1	9.95
	1006	89-WRE-Q	Hicut chain saw, 16 in.	1	256.99
	1007	13-Q2/P2	7.25-in. pwr. saw blade	2	14.99
	1007	54778-2T	Rat-tail file, 1/8-in. fine	1	4.99
	1008	PVC23DRT	PVC pipe, 3.5-in., 8-ft	5	5.87
	1008	WR3/TT3	Steel matting, 4'x8'x1/...	3	119.95
	1008	23109-HB	Claw hammer	1	9.95

FIGURE 8.3 JOIN ON RESULTS

The screenshot shows a Windows application window titled "SQL Plus". Inside, an SQL query is run against three tables: INVOICE, LINE, and PRODUCT. The query joins INVOICE and LINE on their common column INV_NUMBER, and then joins LINE and PRODUCT on their common column P_CODE. The resulting data is displayed in a tabular format with columns: INV_NUMBER, P_CODE, P_DESCRPT, LINE_UNITS, and LINE_PRICE. The output shows 18 rows of product information with their respective quantities and prices.

INV_NUMBER	P_CODE	P_DESCRPT	LINE_UNITS	LINE_PRICE
1001	13-Q2/P2	7.25-in. pwr. saw blade	1	14.99
1001	23109-HB	Claw hammer	1	9.95
1002	54778-2T	Rat-tail file, 1/8-in. fine	2	4.99
1003	2238/QPD	B&D cordless drill, 1/2-in.	1	38.95
1003	1546-QQ2	Hrd. cloth, 1/4-in., 2x50	1	39.95
1003	13-Q2/P2	7.25-in. pwr. saw blade	5	14.99
1004	54778-2T	Rat-tail file, 1/8-in. fine	3	4.99
1004	23109-HB	Claw hammer	2	9.95
1005	PVC23DRT	PVC pipe, 3.5-in., 8-ft	12	5.87
1006	SM-18277	1.25-in. metal screw, 25	3	6.99
1006	2232/QTY	B&D jigsaw, 12-in. blade	1	109.92
1006	23109-HB	Claw hammer	1	9.95
1006	89-WRE-Q	Hicut chain saw, 16 in.	1	256.99
1007	13-Q2/P2	7.25-in. pwr. saw blade	2	14.99
1007	54778-2T	Rat-tail file, 1/8-in. fine	1	4.99
1008	PVC23DRT	PVC pipe, 3.5-in., 8-ft	5	5.87
1008	WR3/TT3	Steel matting, 4'x8'x1/6", .5" mesh	3	119.95
1008	23109-HB	Claw hammer	1	9.95

18 rows selected.

SQL> -

FIGURE 8.4 LEFT JOIN RESULTS

The screenshot shows an SQL Plus window with the title "SQL Plus". The window displays the results of a SQL query. The query is:

```
SQL> SELECT P_CODE, VENDOR.V_CODE, V_NAME
  2  FROM VENDOR LEFT JOIN PRODUCT ON VENDOR.V_CODE = PRODUCT.V_CODE;
```

The results are displayed in a tabular format:

P_CODE	V_CODE	V_NAME
11QER/31	25595	Rubicon Systems
13-Q2/P2	21344	Gomez Bros.
14-Q1/L3	21344	Gomez Bros.
1546-QQ2	23119	Randsets Ltd.
1558-QW1	23119	Randsets Ltd.
2232/QTY	24288	ORDVA, Inc.
2232/QWE	24288	ORDVA, Inc.
2238/QPD	25595	Rubicon Systems
23109-HB	21225	Bryson, Inc.
54778-2T	21344	Gomez Bros.
89-WRE-Q	24288	ORDVA, Inc.
SM-18277	21225	Bryson, Inc.
SW-23116	21231	D&E Supply
WR3/TT3	25595	Rubicon Systems
	22567	Dome Supply
	21226	SuperLoo, Inc.
	24004	Brackman Bros.
	25501	Damal Supplies
	25443	B&K, Inc.

Below the table, the message "19 rows selected." is displayed. At the bottom of the window, the prompt "SQL> -" is visible.

FIGURE 8.5 RIGHT JOIN RESULTS

The screenshot shows a window titled "SQL Plus" displaying the results of a SQL query. The query is:

```
SQL> SELECT P_CODE, VENDOR.V_CODE, V_NAME  
2  FROM VENDOR RIGHT JOIN PRODUCT ON VENDOR.V_CODE = PRODUCT.V_CODE;
```

The output shows the following data:

P_CODE	V_CODE	V_NAME
SM-18277	21225	Bryson, Inc.
23109-HB	21225	Bryson, Inc.
SW-23116	21231	D&E Supply
54778-2T	21344	Gomez Bros.
14-Q1/L3	21344	Gomez Bros.
13-Q2/P2	21344	Gomez Bros.
1558-QW1	23119	Randsets Ltd.
1546-QQ2	23119	Randsets Ltd.
89-WRE-Q	24288	ORDVA, Inc.
2232/QWE	24288	ORDVA, Inc.
2232/QTY	24288	ORDVA, Inc.
WR3/TT3	25595	Rubicon Systems
2238/QPD	25595	Rubicon Systems
11QER/31	25595	Rubicon Systems
PVC23DRT		
23114-AA		

16 rows selected.

SQL> -

FIGURE 8.6 FULL JOIN RESULTS

The screenshot shows an SQL Plus window with a light blue header bar containing the title "SQL Plus". The main area displays the results of a SQL query. The query is:

```
SQL> SELECT P_CODE, VENDOR.V_CODE, V_NAME  
2  FROM VENDOR FULL JOIN PRODUCT ON VENDOR.V_CODE = PRODUCT.V_CODE;
```

The output is a table with three columns: P_CODE, V_CODE, and V_NAME. The data is as follows:

P_CODE	V_CODE	V_NAME
11QER/31	25595	Rubicon Systems
13-Q2/P2	21344	Gomez Bros.
14-Q1/L3	21344	Gomez Bros.
1546-QQ2	23119	Randsets Ltd.
1558-QW1	23119	Randsets Ltd.
2232/QTY	24288	ORDVA, Inc.
2232/QWE	24288	ORDVA, Inc.
2238/QPD	25595	Rubicon Systems
23109-HB	21225	Bryson, Inc.
23114-AA		
54778-2T	21344	Gomez Bros.
89-WRE-Q	24288	ORDVA, Inc.
PVC23DRT		
SM-18277	21225	Bryson, Inc.
SW-23116	21231	D&E Supply
WR3/TT3	25595	Rubicon Systems
	22567	Dome Supply
	21226	SuperLoo, Inc.
	24004	Brackman Bros.
	25501	Damal Supplies
	25443	B&K, Inc.

At the bottom of the output, it says "21 rows selected." followed by the SQL prompt "SQL>".

Subqueries and Correlated Queries

- Subquery is a query inside another query
- Subquery can return:
 - One single value - One column and one row
 - A list of values - One column and multiple rows
 - A virtual table - Multicolumn, multirow set of values
 - No value - Output of the outer query might result in an error or a null empty set

WHERE Subqueries

- Uses inner SELECT subquery on the right side of a WHERE comparison expression
- Value generated by the subquery must be of a comparable data type
- If the query returns more than a single value, the DBMS will generate an error
- Can be used in combination with joins

FIGURE 8.7 WHERE SUBQUERY EXAMPLES

The screenshot shows an SQL Plus window with the title bar "SQL Plus". Inside, two SQL queries are run and their results are displayed.

```
SQL> SELECT P_CODE, P_PRICE FROM PRODUCT
  2 WHERE P_PRICE >= (SELECT AVG(P_PRICE) FROM PRODUCT);

  P_CODE          P_PRICE
  -----          -----
11QER/31        109.99
2232/QTY        109.92
2232/QWE        99.87
89-WRE-Q        256.99
WR3/TT3         119.95

SQL> SELECT DISTINCT CUS_CODE, CUS_LNAME, CUS_FNAME
  2 FROM CUSTOMER JOIN INVOICE USING (CUS_CODE)
  3           JOIN LINE USING (INV_NUMBER)
  4           JOIN PRODUCT USING (P_CODE)
  5 WHERE P_CODE = (SELECT P_CODE FROM PRODUCT WHERE P_DESCRIP = 'Claw hammer');

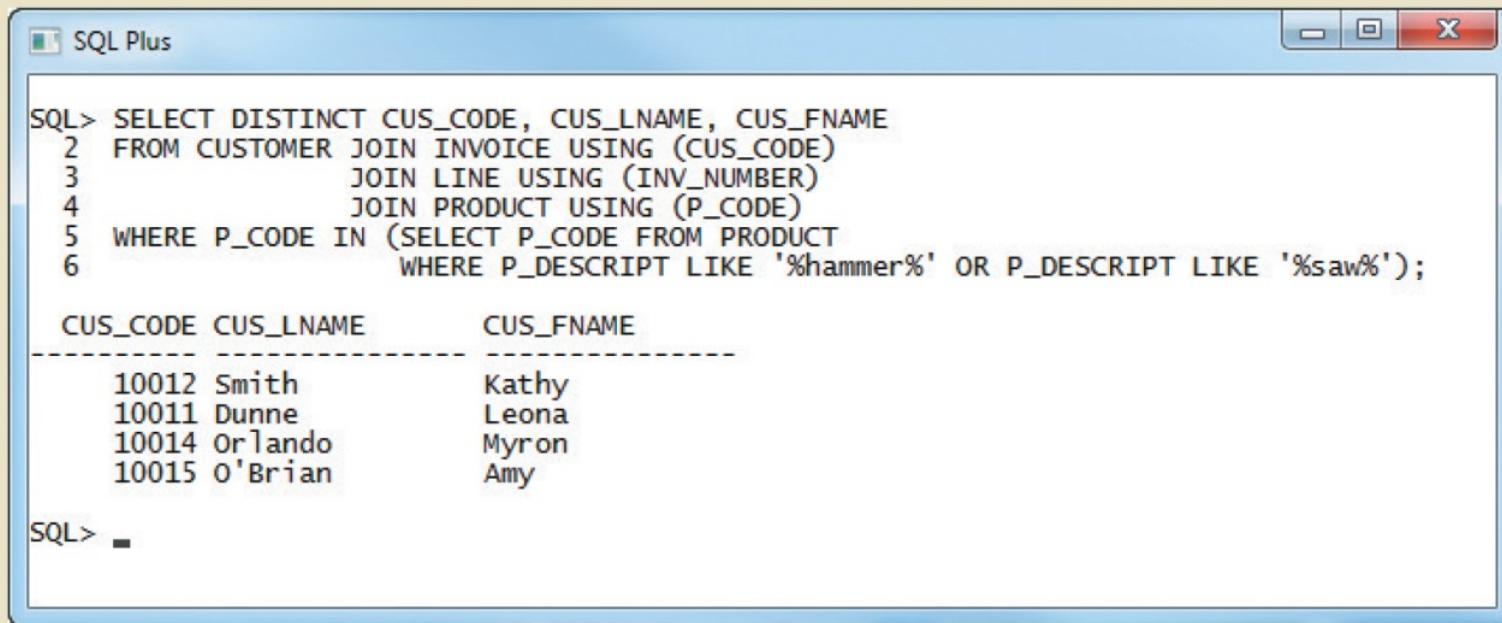
  CUS_CODE CUS_LNAME      CUS_FNAME
  -----  -----
    10011 Dunne            Leona
    10014 Orlando          Myron

SQL> ■
```

IN and HAVING Subqueries

- IN subqueries
 - Used to compare a single attribute to a list of values
- HAVING subqueries
 - HAVING clause restricts the output of a GROUP BY query by applying conditional criteria to the grouped rows

FIGURE 8.8 IN SUBQUERY EXAMPLE



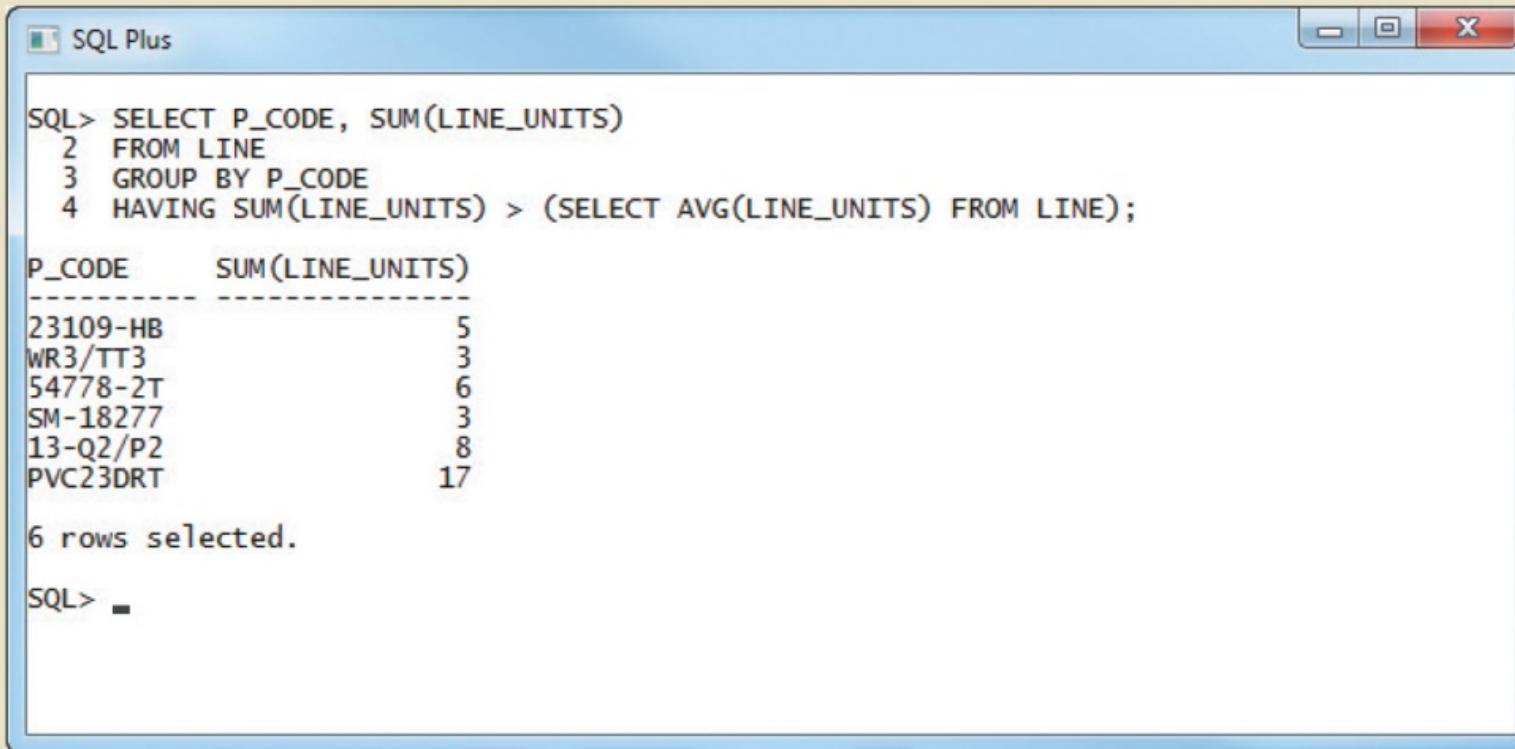
The screenshot shows a Windows-style window titled "SQL Plus". Inside, an SQL query is displayed and executed:

```
SQL> SELECT DISTINCT CUS_CODE, CUS_LNAME, CUS_FNAME
  2  FROM CUSTOMER JOIN INVOICE USING (CUS_CODE)
  3          JOIN LINE USING (INV_NUMBER)
  4          JOIN PRODUCT USING (P_CODE)
  5 WHERE P_CODE IN (SELECT P_CODE FROM PRODUCT
  6                   WHERE P_DESCRIP LIKE '%hammer%' OR P_DESCRIP LIKE '%saw%'));

  CUS_CODE CUS_LNAME      CUS_FNAME
  -----  -----
    10012  Smith          Kathy
    10011  Dunne          Leona
    10014  Orlando        Myron
    10015  O'Brian        Amy
```

The query selects distinct customer codes, last names, and first names from the CUSTOMER table, joining it with the INVOICE, LINE, and PRODUCT tables. It filters the results using an IN subquery that checks if the product code exists in the PRODUCT table where the product description contains either "%hammer%" or "%saw%". The output shows four customers: Smith, Dunne, Orlando, and O'Brian.

FIGURE 8.9 HAVING SUBQUERY EXAMPLE



The screenshot shows a window titled "SQL Plus". Inside, a SQL query is run:

```
SQL> SELECT P_CODE, SUM(LINE_UNITS)
  2  FROM LINE
  3  GROUP BY P_CODE
  4  HAVING SUM(LINE_UNITS) > (SELECT AVG(LINE_UNITS) FROM LINE);
```

The output displays the results of the query:

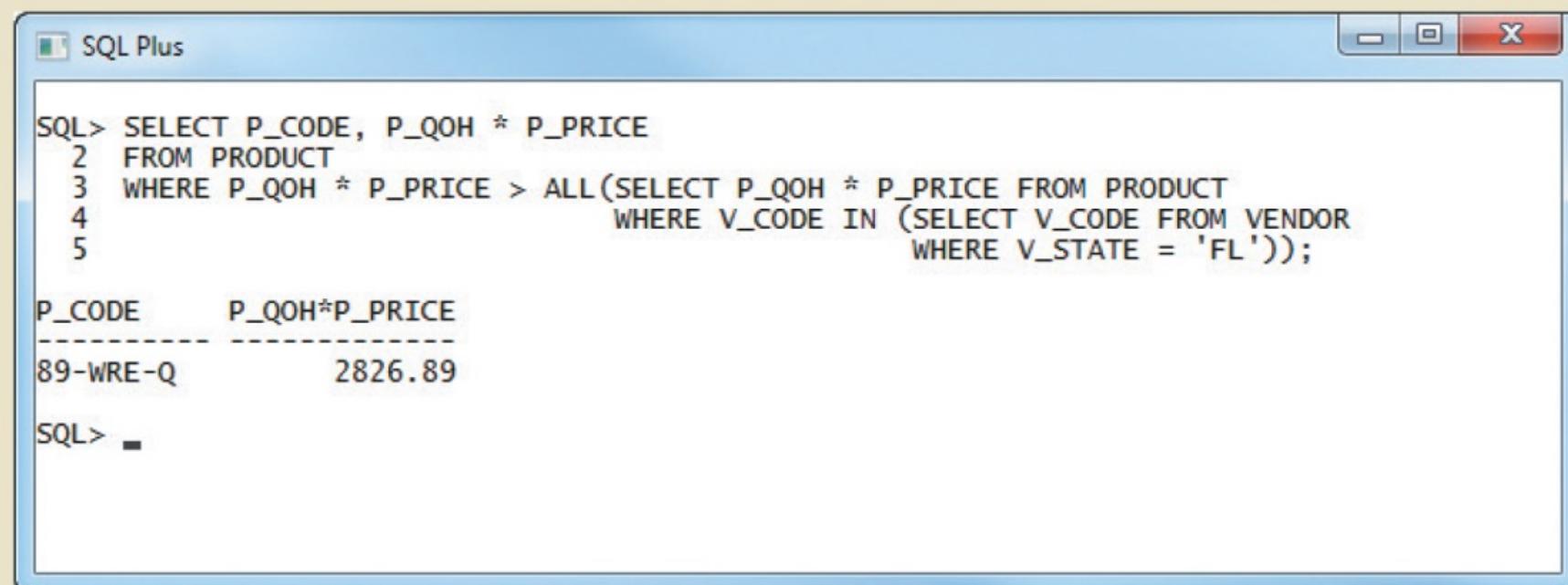
P_CODE	SUM(LINE_UNITS)
23109-HB	5
WR3/TT3	3
54778-2T	6
SM-18277	3
13-Q2/P2	8
PVC23DRT	17

Below the table, the message "6 rows selected." is shown. At the bottom, there is a prompt "SQL> -".

Multirow Subquery Operators: ANY and ALL

- ALL operator
 - Allows comparison of a single value with a list of values returned by the first subquery
 - Uses a comparison operator other than equals
- ANY operator
 - Allows comparison of a single value to a list of values and selects only the rows for which the value is greater than or less than any value in the list

FIGURE 8.10 MULTIROW SUBQUERY OPERATOR EXAMPLE



The screenshot shows an SQL Plus window with the title "SQL Plus". The window contains the following SQL code and its execution results:

```
SQL> SELECT P_CODE, P_QOH * P_PRICE
  2  FROM PRODUCT
  3 WHERE P_QOH * P_PRICE > ALL(SELECT P_QOH * P_PRICE FROM PRODUCT
  4                               WHERE V_CODE IN (SELECT V_CODE FROM VENDOR
  5                                 WHERE V_STATE = 'FL'));
```

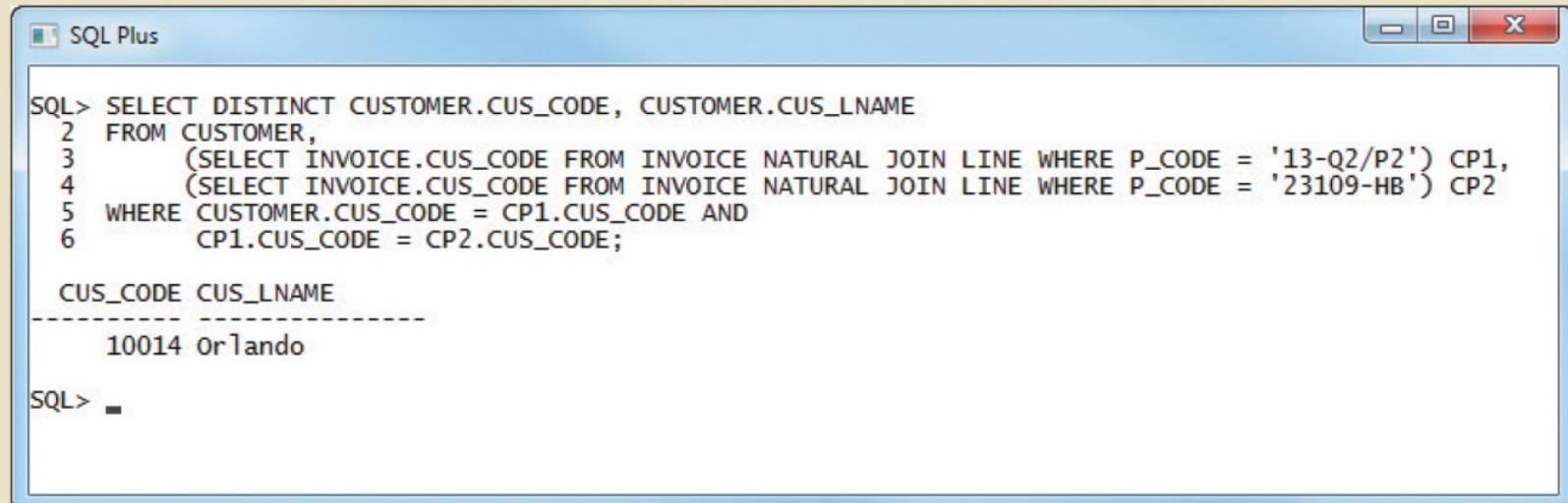
P_CODE	P_QOH*P_PRICE
89-WRE-Q	2826.89

SQL> -

FROM and Attribute List Subqueries

- FROM clause:
 - Specifies the tables from which the data will be drawn
 - Can use SELECT subquery
- SELECT statement uses attribute list to indicate what columns to project in the resulting set
- Inline subquery
 - Subquery expression included in the attribute list that must return one value
- Column alias cannot be used in attribute list computation if alias is defined in the same attribute list

FIGURE 8.11 FROM SUBQUERY EXAMPLE



The screenshot shows a Windows application window titled "SQL Plus". Inside the window, an SQL query is displayed and executed. The query uses a FROM clause with two subqueries. The first subquery selects CUS_CODE from CUSTOMER where the P_CODE is '13-Q2/P2'. The second subquery selects CUS_CODE from CUSTOMER where the P_CODE is '23109-HB'. These results are joined with the main CUSTOMER table to find customers who have purchased both items. The output shows one customer with CUS_CODE 10014 and LNAME Orlando.

```
SQL> SELECT DISTINCT CUSTOMER.CUS_CODE, CUSTOMER.CUS_LNAME
  2  FROM CUSTOMER,
  3      (SELECT INVOICE.CUS_CODE FROM INVOICE NATURAL JOIN LINE WHERE P_CODE = '13-Q2/P2') CP1,
  4      (SELECT INVOICE.CUS_CODE FROM INVOICE NATURAL JOIN LINE WHERE P_CODE = '23109-HB') CP2
  5 WHERE CUSTOMER.CUS_CODE = CP1.CUS_CODE AND
  6       CP1.CUS_CODE = CP2.CUS_CODE;

  CUS_CODE CUS_LNAME
  -----
    10014 Orlando

SQL> -
```

FIGURE 8.12 INLINE SUBQUERY EXAMPLE

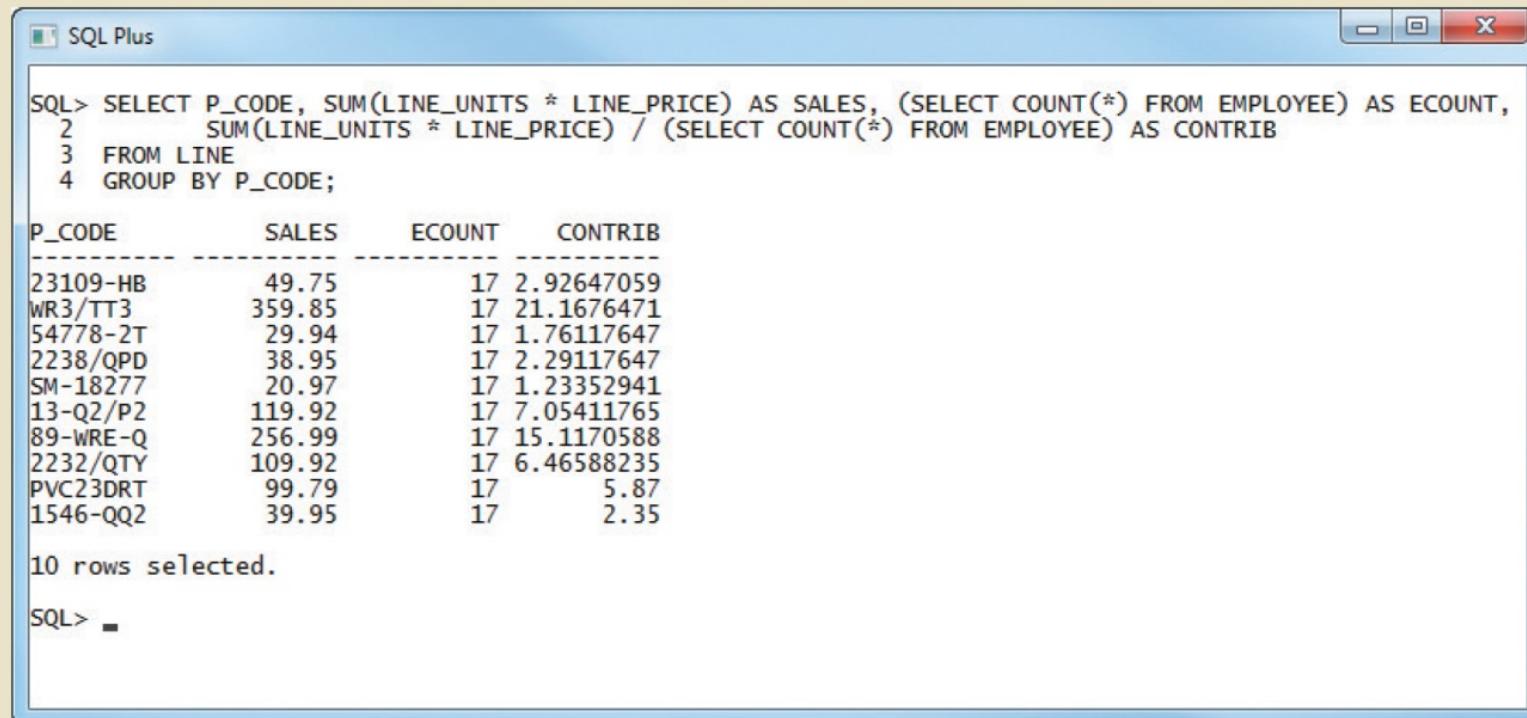
The screenshot shows an SQL query being run in a database environment. The query is:

```
1 • SELECT P_CODE, P_PRICE, (SELECT AVG(P_PRICE) FROM PRODUCT) AS AVGPRICE,
2          P_PRICE - (SELECT AVG(P_PRICE) FROM PRODUCT) AS DIFF
3   FROM PRODUCT;
```

The results are displayed in a grid format:

	P_CODE	F_PRICE	AVGPRICE	DIFF
▶	11QER/31	109.99	56.4212492108345	53.56874865293503
	13-Q2/P2	14.99	56.4212492108345	-41.43124943971634
	14-Q1/L3	17.49	56.4212492108345	-38.93124943971634
	1546-QQ2	39.95	56.4212492108345	-16.47124844789505
	1558-QW1	43.99	56.4212492108345	-12.431247532367706
	2232/QTY	109.92	56.4212492108345	53.49874895811031
	2232/QWE	99.87	56.4212492108345	43.44875353574753
	2238/QPD	38.95	56.4212492108345	-17.47124844789505
	23109-HB	9.95	56.4212492108345	-46.47124940156937
	23114-AA	14.4	56.4212492108345	-42.02124959230423
	54778-2T	4.99	56.4212492108345	-51.43124943971634
	89-WRE-Q	256.99	56.4212492108345	200.5687410235405
	PVC23DRT	5.87	56.4212492108345	-50.55124932527542
	SM-18277	6.99	56.4212492108345	-49.43124943971634
	SW-23116	8.45	56.4212492108345	-47.97124940156937
	WR3/TT3	119.95	56.4212492108345	63.528747737407684

FIGURE 8.13 ANOTHER EXAMPLE OF AN INLINE SUBQUERY



The screenshot shows an SQL Plus window with the title bar "SQL Plus". The window contains the following SQL code and its execution results:

```
SQL> SELECT P_CODE, SUM(LINE_UNITS * LINE_PRICE) AS SALES, (SELECT COUNT(*) FROM EMPLOYEE) AS ECOUNT,
2      SUM(LINE_UNITS * LINE_PRICE) / (SELECT COUNT(*) FROM EMPLOYEE) AS CONTRIB
3  FROM LINE
4 GROUP BY P_CODE;
```

P_CODE	SALES	ECOUNT	CONTRIB
23109-HB	49.75	17	2.92647059
WR3/TT3	359.85	17	21.1676471
54778-2T	29.94	17	1.76117647
2238/QPD	38.95	17	2.29117647
SM-18277	20.97	17	1.23352941
13-Q2/P2	119.92	17	7.05411765
89-WRE-Q	256.99	17	15.1170588
2232/QTY	109.92	17	6.46588235
PVC23DRT	99.79	17	5.87
1546-QQ2	39.95	17	2.35

10 rows selected.

SQL> -

Correlated Subqueries

- Executes once for each row in the outer query
- Inner query references a column of the outer subquery
- Can be used with the EXISTS special operator

FIGURE 8.14 CORRELATED SUBQUERY EXAMPLES

The screenshot shows a Windows application window titled "SQL Plus". Inside, two SQL queries are run against a database named "LINE".

Query 1:

```
SQL> SELECT INV_NUMBER, P_CODE, LINE_UNITS
  2  FROM LINE LS
  3  WHERE LS.LINE_UNITS > (SELECT AVG(LINE_UNITS)
  4                                FROM LINE LA
  5                                WHERE LA.P_CODE = LS.P_CODE);
```

Output:

INV_NUMBER	P_CODE	LINE_UNITS
1003	13-Q2/P2	5
1004	54778-2T	3
1004	23109-HB	2
1005	PVC23DRT	12

Query 2:

```
SQL> SELECT INV_NUMBER, P_CODE, LINE_UNITS,
  2          (SELECT AVG(LINE_UNITS) FROM LINE LX WHERE LX.P_CODE = LS.P_CODE) AS AVG
  3  FROM LINE LS
  4  WHERE LS.LINE_UNITS > (SELECT AVG(LINE_UNITS)
  5                                FROM LINE LA
  6                                WHERE LA.P_CODE = LS.P_CODE);
```

Output:

INV_NUMBER	P_CODE	LINE_UNITS	AVG
1004	23109-HB	2	1.25
1004	54778-2T	3	2
1003	13-Q2/P2	5	2.66666667
1005	PVC23DRT	12	8.5

SQL> -

FIGURE 8.15 EXISTS CORRELATED SUBQUERY EXAMPLES

The screenshot shows a window titled "SQL Plus" with two separate SQL queries and their results.

```
SQL> SELECT CUS_CODE, CUS_LNAME, CUS_FNAME
  2  FROM CUSTOMER
  3 WHERE EXISTS (SELECT CUS_CODE FROM INVOICE
  4                   WHERE INVOICE.CUS_CODE = CUSTOMER.CUS_CODE);

  CUS_CODE CUS_LNAME      CUS_FNAME
  -----  -----
    10014  Orlando        Myron
    10011  Dunne          Leona
    10012  Smith          Kathy
    10018  Farriss        Anne
    10015  O'Brian        Amy

SQL> SELECT V_CODE, V_NAME FROM VENDOR
  2 WHERE EXISTS (SELECT * FROM PRODUCT
  3                   WHERE P_QOH < P_MIN * 2 AND VENDOR.V_CODE = PRODUCT.V_CODE);

  V_CODE V_NAME
  -----  -----
    25595 Rubicon Systems
    21344 Gomez Bros.
    23119 Randsets Ltd.
    24288 ORDVA, Inc.

SQL> -
```

SQL Functions

- Functions always use a numerical, date, or string value
- Value may be part of a command or may be an attribute located in a table
- Function may appear anywhere in an SQL statement where a value or an attribute can be used

SQL Functions

Date and time functions

Numeric functions

String functions

Conversion functions

Relational Set Operators

- SQL data manipulation commands are set-oriented
 - **Set-oriented:** Operate over entire sets of rows and columns at once
- UNION, INTERSECT, and Except (MINUS) work properly when relations are union-compatible
 - **Union-compatible:** Number of attributes are the same and their corresponding data types are alike
- UNION
 - Combines rows from two or more queries without including duplicate rows

```

SELECT CUS_LNAME, CUS_FNAME, CUS_INITIAL, CUS_AREACODE, CUS_PHONE
FROM CUSTOMER
UNION
SELECT CUS_LNAME, CUS_FNAME, CUS_INITIAL, CUS_AREACODE, CUS_PHONE
FROM CUSTOMER_2

```

FIGURE 8.16 UNION QUERY RESULTS

Database name: Ch08_SaleCo

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	345.86
10013	Ołowski	Paul	F	615	894-2180	536.75
10014	Orlando	Myron		615	222-1672	0.00
10015	O'Brian	Amy	B	713	442-3381	0.00
10016	Brown	James	G	615	297-1228	221.19
10017	Williams	George		615	290-2556	768.93
10018	Farriss	Anne	G	713	382-7185	216.55
10019	Smith	Olette	K	615	297-3809	0.00

Query name: qryUNION-of-CUSTOMER-and-CUSTOMER_2

CUS_LNAME	CUS_FNAME	CUS_INITIAL	CUS_AREACODE	CUS_PHONE
Brown	James	G	615	297-1228
Dunne	Leona	K	713	894-1238
Farriss	Anne	G	713	382-7185
Hernandez	Carlos	J	723	123-7654
Lewis	Marie	J	734	332-1789
McDowell	George		723	123-7768
O'Brian	Amy	B	713	442-3381
Ołowski	Paul	F	615	894-2180
Orlando	Myron		615	222-1672
Ramas	Alfred	A	615	844-2573
Smith	Kathy	W	615	894-2285
Smith	Olette	K	615	297-3809
Terrell	Justine	H	615	322-9870
Tirpin	Khaled	G	723	123-9876
Williams	George		615	290-2556

Table name: CUSTOMER_2

CUS_CODE	CUS_LNAME	CUS_FNAME	CUS_INITIAL	CUS_AREACODE	CUS_PHONE
345	Terrell	Justine	H	615	322-9870
347	Ołowski	Paul	F	615	894-2180
351	Hernandez	Carlos	J	723	123-7654
352	McDowell	George		723	123-7768
365	Tirpin	Khaled	G	723	123-9876
368	Lewis	Marie	J	734	332-1789
369	Dunne	Leona	K	713	894-1238

Relational Set Operators

- Syntax - query UNION query
- UNION ALL
 - Produces a relation that retains duplicate rows
 - Can be used to unite more than two queries
- INTERSECT
 - Combines rows from two queries, returning only the rows that appear in both sets
 - Syntax - query INTERSECT query

```

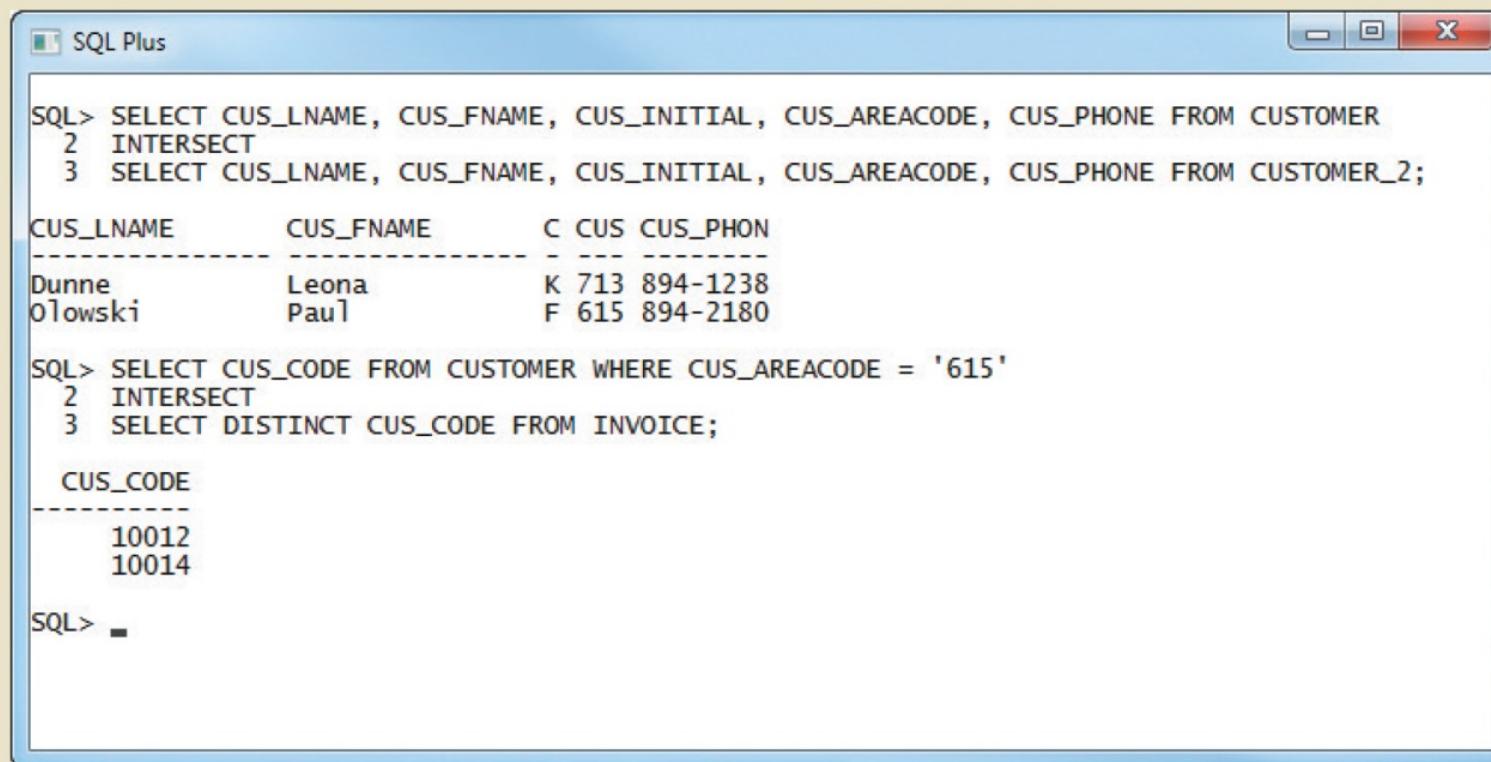
SELECT CUS_LNAME, CUS_FNAME, CUS_INITIAL, CUS_AREACODE, CUS_PHONE
FROM CUSTOMER
UNION ALL
SELECT CUS_LNAME, CUS_FNAME, CUS_INITIAL, CUS_AREACODE, CUS_PHONE
FROM CUSTOMER_2

```

FIGURE 8.17 UNION ALL QUERY RESULTS

Database name: Ch08_SaleCo						
Table name: CUSTOMER			Query name: qryUNION-ALL-of-CUSTOMER-and-CUSTOMER_2			
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	345.86
10013	Olowksi	Paul	F	615	894-2180	536.75
10014	Orlando	Myron		615	222-1672	0.00
10015	O'Brian	Amy	B	713	442-3381	0.00
10016	Brown	James	G	615	297-1228	221.19
10017	Williams	George		615	290-2556	768.93
10018	Farris	Anne	G	713	382-7185	216.55
10019	Smith	Olette	K	615	297-3809	0.00
Table name: CUSTOMER_2						
CUS_CODE	CUS_LNAME	CUS_FNAME	CUS_INITIAL	CUS_AREACODE	CUS_PHONE	
345	Terrill	Justine	H	615	322-9870	
347	Olowksi	Paul	F	615	894-2180	
351	Hernandez	Carlos	J	723	123-7654	
352	McDowell	George		723	123-7768	
365	Tirpin	Khaleed	G	723	123-9876	
368	Lewis	Marie	J	734	332-1789	
369	Dunne	Leona	K	713	894-1238	

FIGURE 8.18 INTERSECT QUERY RESULTS



The screenshot shows an SQL Plus window with the title bar "SQL Plus". The window contains two separate SQL queries. The first query retrieves customer names and phone numbers from two tables: CUSTOMER and CUSTOMER_2, using the INTERSECT operator. The second query retrieves customer codes from the CUSTOMER table where the area code is 615, and then finds distinct customer codes from the INVOICE table using the INTERSECT operator.

```
SQL> SELECT CUS_LNAME, CUS_FNAME, CUS_INITIAL, CUS_AREACODE, CUS_PHONE FROM CUSTOMER
  2 INTERSECT
  3 SELECT CUS_LNAME, CUS_FNAME, CUS_INITIAL, CUS_AREACODE, CUS_PHONE FROM CUSTOMER_2;
CUS_LNAME      CUS_FNAME      CUS_INITIAL CUS_AREACODE CUS_PHONE
-----        -----        -----       -----
Dunne          Leona          K            713         894-1238
Olowski        Paul           F            615         894-2180

SQL> SELECT CUS_CODE FROM CUSTOMER WHERE CUS_AREACODE = '615'
  2 INTERSECT
  3 SELECT DISTINCT CUS_CODE FROM INVOICE;
CUS_CODE
-----
10012
10014

SQL> ■
```

Relational Set Operators

- EXCEPT (MINUS)
 - Combines rows from two queries and returns only the rows that appear in the first set
 - Syntax
 - query EXCEPT query
 - query MINUS query
- Syntax alternatives
 - IN and NOT IN subqueries can be used in place of INTERSECT

FIGURE 8.19 MINUS QUERY RESULTS

The screenshot shows an SQL Plus window with three distinct minus query operations. The first operation compares the CUSTOMER and CUSTOMER_2 tables. The second operation compares the CUSTOMER_2 and CUSTOMER tables. The third operation filters the CUSTOMER table by area code 615 and then subtracts the distinct codes from the INVOICE table.

```
SQL> SELECT CUS_LNAME, CUS_FNAME, CUS_INITIAL, CUS_AREACODE, CUS_PHONE FROM CUSTOMER
  2 MINUS
  3 SELECT CUS_LNAME, CUS_FNAME, CUS_INITIAL, CUS_AREACODE, CUS_PHONE FROM CUSTOMER_2;
CUS_LNAME      CUS_FNAME      C CUS CUS_PHON
-----  -----
Brown          James          G 615 297-1228
Farriss        Anne           G 713 382-7185
O'Brian        Amy            B 713 442-3381
Orlando         Myron          615 222-1672
Ramas           Alfred          A 615 844-2573
Smith           Kathy           W 615 894-2285
Smith           Olette          K 615 297-3809
Williams        George          615 290-2556

8 rows selected.

SQL> SELECT CUS_LNAME, CUS_FNAME, CUS_INITIAL, CUS_AREACODE, CUS_PHONE FROM CUSTOMER_2
  2 MINUS
  3 SELECT CUS_LNAME, CUS_FNAME, CUS_INITIAL, CUS_AREACODE, CUS_PHONE FROM CUSTOMER;
CUS_LNAME      CUS_FNAME      C CUS CUS_PHON
-----  -----
Hernandez      Carlos          J 723 123-7654
Lewis           Marie           J 734 332-1789
McDowell       George          723 123-7768
Terrell         Justine         H 615 322-9870
Tirpin          Khaleed         G 723 123-9876

SQL> SELECT CUS_CODE FROM CUSTOMER WHERE CUS_AREACODE = '615'
  2 MINUS
  3 SELECT DISTINCT CUS_CODE FROM INVOICE;
CUS_CODE
-----
10010
10013
10016
10017
10019

SQL> -
```

```

SELECT CUS_CODE
FROM CUSTOMER
WHERE CUS_AREACODE = '615' AND
CUS_CODE IN (SELECT DISTINCT CUS_CODE FROM INVOICE)

```

FIGURE 8.20 INTERSECT ALTERNATIVE

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	345.86
10013	Ołowski	Paul	F	615	894-2180	536.75
10014	Orlando	Myron		615	222-1672	0.00
10015	O'Brian	Amy	B	713	442-3381	0.00
10016	Brown	James	G	615	297-1228	221.19
10017	Williams	George		615	290-2556	768.93
10018	Farris	Anne	G	713	382-7185	216.55
10019	Smith	Olette	K	615	297-3809	0.00

Database name: Ch08_SaleCo

Table name: INVOICE

INV_NUMBER	CUS_CODE	INV_DATE
1001	10014	16-Jan-14
1002	10011	16-Jan-14
1003	10012	16-Jan-14
1004	10011	17-Jan-14
1005	10018	17-Jan-14
1006	10014	17-Jan-14
1007	10015	17-Jan-14
1008	10011	17-Jan-14

Query name: qry-INTERSECT-Alternative

CUS_CODE
10012
10014

```

SELECT CUS_CODE
FROM CUSTOMER
WHERE CUS_AREACODE = '615' AND
CUS_CODE NOT IN (SELECT DISTINCT CUS_CODE FROM INVOICE)

```

FIGURE 8.21 MINUS ALTERNATIVE

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	345.86
10013	Ołowski	Paul	F	615	894-2180	536.75
10014	Orlando	Myron		615	222-1672	0.00
10015	O'Brian	Amy	B	713	442-3381	0.00
10016	Brown	James	G	615	297-1228	221.19
10017	Williams	George		615	290-2556	768.93
10018	Farris	Anne	G	713	382-7185	216.55
10019	Smith	Olette	K	615	297-3809	0.00

Database name: Ch08_SaleCo

Table name: INVOICE

INV_NUMBER	CUS_CODE	INV_DATE
1001	10014	16-Jan-14
1002	10011	16-Jan-14
1003	10012	16-Jan-14
1004	10011	17-Jan-14
1005	10018	17-Jan-14
1006	10014	17-Jan-14
1007	10015	17-Jan-14
1008	10011	17-Jan-14

Query name: qry-MINUS-Alternative

CUS_CODE
10010
10013
10016
10017
10019

Virtual Tables: Creating a View

- **View:** Virtual table based on a SELECT query
- **Base tables:** Tables on which the view is based
- **CREATE VIEW** statement: Data definition command that stores the subquery specification in the data dictionary
 - CREATE VIEW command
 - CREATE VIEW viewname AS SELECT query

FIGURE 8.22 CREATING A VIRTUAL TABLE WITH THE CREATE VIEW COMMAND

The screenshot shows a Windows application window titled "SQL Plus". Inside the window, SQL commands are entered and executed. The output displays the creation of a view named "PRICEGT50" and its contents.

```
SQL> CREATE VIEW PRICEGT50 AS
  2      SELECT P_DESCRPT, P_QOH, P_PRICE
  3      FROM PRODUCT
  4     WHERE P_PRICE > 50;

View created.

SQL> SELECT * FROM PRICEGT50;

P_DESCRPT                  P_QOH    P_PRICE
-----                      --
Power painter, 15 psi., 3-nozzle      8      109.99
B&D jigsaw, 12-in. blade             8      109.92
B&D jigsaw, 8-in. blade              6      99.87
Hicut chain saw, 16 in.              11     256.99
Steel matting, 4'x8'x1/6", .5" mesh   18     119.95

SQL> -
```

FIGURE 8.23 THE PRODMASTER AND PRODSALES TABLES

Database name: Ch08_UV

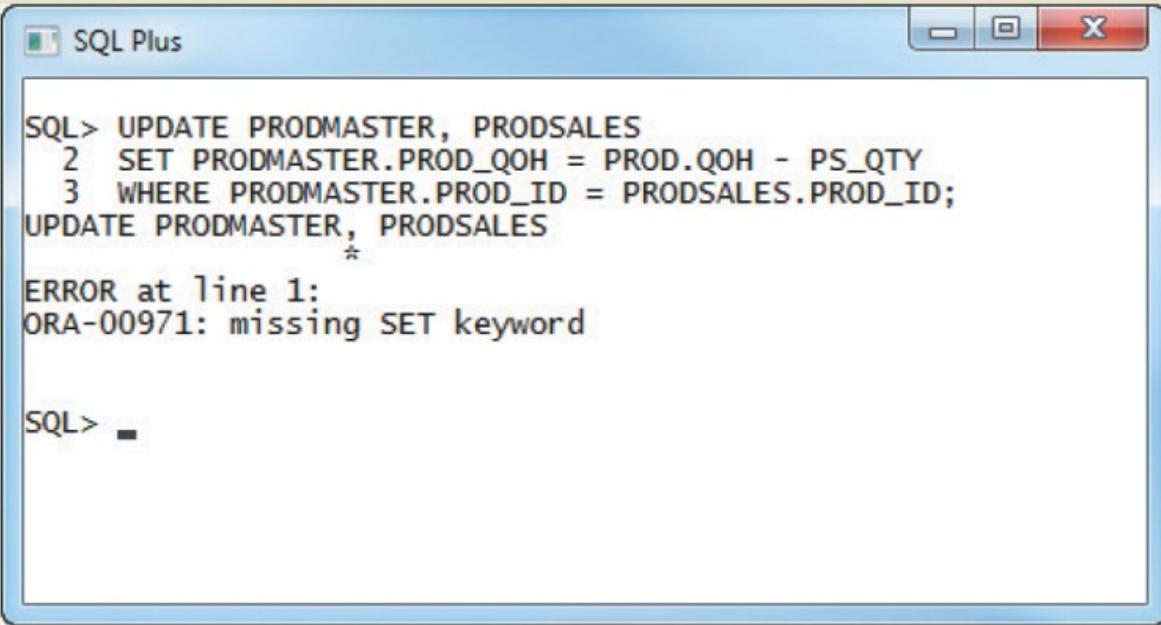
Table name: PRODMASTER

PROD_ID	PROD_DESC	PROD_QOH
A123	SCREWS	67
BX34	NUTS	37
C583	BOLTS	50

Table name: PRODSALES

PROD_ID	PS_QTY
A123	7
BX34	3

FIGURE 8.24 THE ORACLE UPDATE ERROR MESSAGE



The screenshot shows a Windows-style application window titled "SQL Plus". Inside the window, there is a command-line interface where a user has entered the following SQL code:

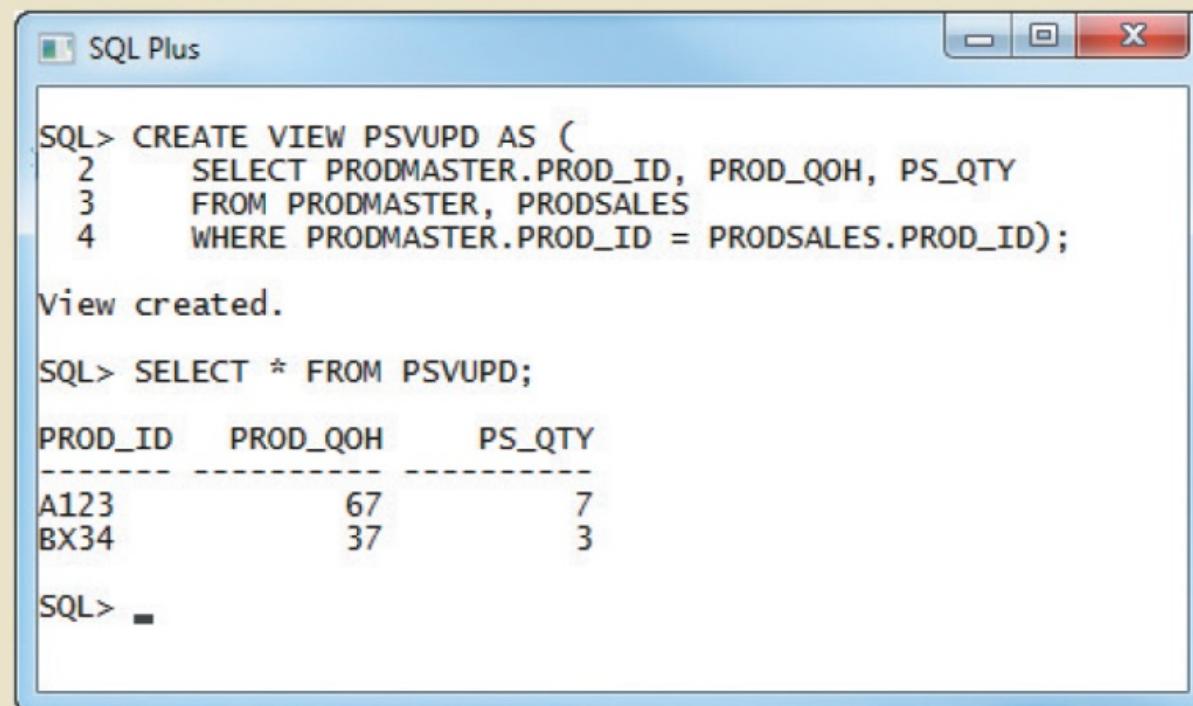
```
SQL> UPDATE PRODMASTER, PRODSALES
  2 SET PRODMASTER.PROD_QOH = PROD.QOH - PS_QTY
  3 WHERE PRODMASTER.PROD_ID = PRODSALES.PROD_ID;
UPDATE PRODMASTER, PRODSALES
*
ERROR at Line 1:
ORA-00971: missing SET keyword
```

After the error message, the prompt "SQL>" is visible again, indicating the user can enter more commands.

Updatable Views

- Used to update attributes in any base tables used in the view
- **Batch update routine:** Pools multiple transactions into a single batch to update a master table field in a single operation
- Updatable view restrictions
 - GROUP BY expressions or aggregate functions cannot be used
 - Set operators cannot be used
 - JOINs or group operators cannot be used

FIGURE 8.25 CREATING AN UPDATABLE VIEW



The screenshot shows an SQL Plus window titled "SQL Plus". Inside the window, the following SQL commands are executed:

```
SQL> CREATE VIEW PSVUPD AS (
  2      SELECT PRODMASTER.PROD_ID, PROD_QOH, PS_QTY
  3      FROM PRODMASTER, PRODSALES
  4     WHERE PRODMASTER.PROD_ID = PRODSALES.PROD_ID);
View created.

SQL> SELECT * FROM PSVUPD;
PROD_ID    PROD_QOH    PS_QTY
-----  -----  -----
A123          67          7
BX34          37          3

SQL> ■
```

The window has standard operating system window controls (minimize, maximize, close) at the top right.

FIGURE 8.26 PRODMASTER TABLE UPDATE, USING AN UPDATABLE VIEW

The screenshot shows an SQL Plus window with the following session history:

```
SQL> SELECT * FROM PRODMASTER;
PROD_ID PROD_DESC      PROD_QOH
----- -----
A123    SCREWS          67
BX34    NUTS             37
C583    BOLTS            50

SQL> SELECT * FROM PRODSALES;
PROD_ID      PS_QTY
----- -----
A123                7
BX34                3

SQL> UPDATE PSVUPD
  2 SET PROD_QOH = PROD_QOH - PS_QTY;
2 rows updated.

SQL> SELECT * FROM PRODMASTER;
PROD_ID PROD_DESC      PROD_QOH
----- -----
A123    SCREWS          60
BX34    NUTS             34
C583    BOLTS            50

SQL> -
```

The window title is "SQL Plus". The session starts with a SELECT query from the PRODMASTER table, displaying three rows: A123 (SCREWS, QOH 67), BX34 (NUTS, QOH 37), and C583 (BOLTS, QOH 50). It then runs a SELECT query from the PRODSALES table, showing two rows: A123 (PS_QTY 7) and BX34 (PS_QTY 3). Following this, an UPDATE statement is executed against a view named PSVUPD, which subtracts the PS_QTY value from the PROD_QOH. The message "2 rows updated." is displayed. Finally, another SELECT query is run on the PRODMASTER table, showing the updated values: A123 (SCREWS, QOH 60), BX34 (NUTS, QOH 34), and C583 (BOLTS, QOH 50). The session ends with a final "-".

Oracle Sequences

- Independent object in the database
- Have a name and can be used anywhere a value expected
- Not tied to a table or column
- Generate a numeric value that can be assigned to any column in any table
- Table attribute with an assigned value can be edited and modified

Figure 8.27 - Oracle Sequence

FIGURE 8.27 ORACLE SEQUENCE

The screenshot shows a Windows application window titled "SQL Plus". Inside, SQL commands are run against an Oracle database. The first command creates a sequence named "CUS_CODE_SEQ" starting at 20010 with NOCACHE. The second command creates a sequence named "INV_NUMBER_SEQ" starting at 4010 with NOCACHE. The third command runs a SELECT statement on the "USER_SEQUENCES" view, which lists the created sequences with their properties: both sequences have a minimum value of 1, a maximum value of 1.0000E+28, an increment by 1, and a cache size of 0. The last number for CUS_CODE_SEQ is 20010 and for INV_NUMBER_SEQ is 4010. The partition count and start/keep values are all null.

```
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 PARTITION_COUNT S K
-----  -----  -----  -----  -----  -----  -----  -----
CUS_CODE_SEQ          1 1.0000E+28          1 N N            0        20010          N N
INV_NUMBER_SEQ         1 1.0000E+28          1 N N            0        4010          N N

SQL> =
```

FIGURE 8.28 ORACLE SEQUENCE EXAMPLES

```
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 08-JUL-15

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.

SQL> -
```

Procedural SQL

- Performs a conditional or looping operation by isolating critical code and making all application programs call the shared code
 - Yields better maintenance and logic control
- **Persistent stored module (PSM):** Block of code containing:
 - Standard SQL statements
 - Procedural extensions that is stored and executed at the DBMS server

Procedural SQL

- **Procedural Language SQL (PL/SQL)**
 - Use and storage of procedural code and SQL statements within the database
 - Merging of SQL and traditional programming constructs
- Procedural code is executed as a unit by DBMS when invoked by end user
- End users can use PL/SQL to create:
 - Anonymous PL/SQL blocks and triggers
 - Stored procedures and PL/SQL functions

Table 8.9 - PL/SQL Basic Data Types

TABLE 8.9

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

Triggers

- Procedural SQL code automatically invoked by RDBMS when given data manipulation event occurs
- Parts of a trigger definition
 - Triggering timing - Indicates when trigger's PL/SQL code executes
 - Triggering event - Statement that causes the trigger to execute
 - Triggering level - **Statement-** and **row-level**
 - Triggering action - PL/SQL code enclosed between the BEGIN and END keywords

Triggers

- **DROP TRIGGER trigger_name command**
 - Deletes a trigger without deleting the table
- Trigger action based on DML predicates
 - Actions depend on the type of DML statement that fires the trigger

Stored Procedures

- Named collection of procedural and SQL statements
- Advantages
 - Reduce network traffic and increase performance
 - Reduce code duplication by means of code isolation and code sharing

PL/SQL Processing with Cursors

- **Cursor:** Special construct used to hold data rows returned by a SQL query
- **Implicit cursor:** Automatically created when SQL statement returns only one value
- **Explicit cursor:** Holds the output of a SQL statement that may return two or more rows
- Cursor-style processing involves retrieving data from the cursor one row at a time
 - Current row is copied to PL/SQL variables

Table 8.10 - Cursor Processing Commands

TABLE 8.10

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 does not populate the cursor with the data. Before you can use a cursor, you need to open it. For example:</p> <p><code>OPEN cursor_name</code></p>
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> <p><code>FETCH cursor_name INTO variable1 [, variable2, ...]</code></p> <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	The CLOSE command closes the cursor for processing.

Table 8.11 - Cursor Attributes

TABLE 8.11

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 Stored Functions

- **Stored function:** Named group of procedural and SQL statements that returns a value
 - As indicated by a RETURN statement in its program code
- Can be invoked only from within stored procedures or triggers

Embedded SQL

- SQL statements contained within an application programming language
- **Host language:** Any language that contains embedded SQL statements
- Differences between SQL and procedural languages
 - Run-time mismatch
 - SQL is executed one instruction at a time
 - Host language runs at client side in its own memory space

Embedded SQL

- Processing mismatch
 - Conventional programming languages process one data element at a time
 - Newer programming environments manipulate data sets in a cohesive manner
- Data type mismatch
 - Data types provided by SQL might not match data types used in different host languages

Embedded SQL

- Embedded SQL framework defines:
 - Standard syntax to identify embedded SQL code within the host language
 - Standard syntax to identify host variables
 - Communication area used to exchange status and error information between SQL and host language

Table 8.12 - SQL Status and Error Reporting Variables

TABLE 8.12

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 and 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 XXYYY where: XX-> represents the class code YYY-> represents the subclass code

Embedded SQL

- **Static SQL:** Programmer uses predefined SQL statements and parameters
 - SQL statements will not change while application is running
- **Dynamic SQL:** SQL statement is generated at run time
 - Attribute list and condition are not known until end user specifies them
 - Slower than static SQL
 - Requires more computer resources