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

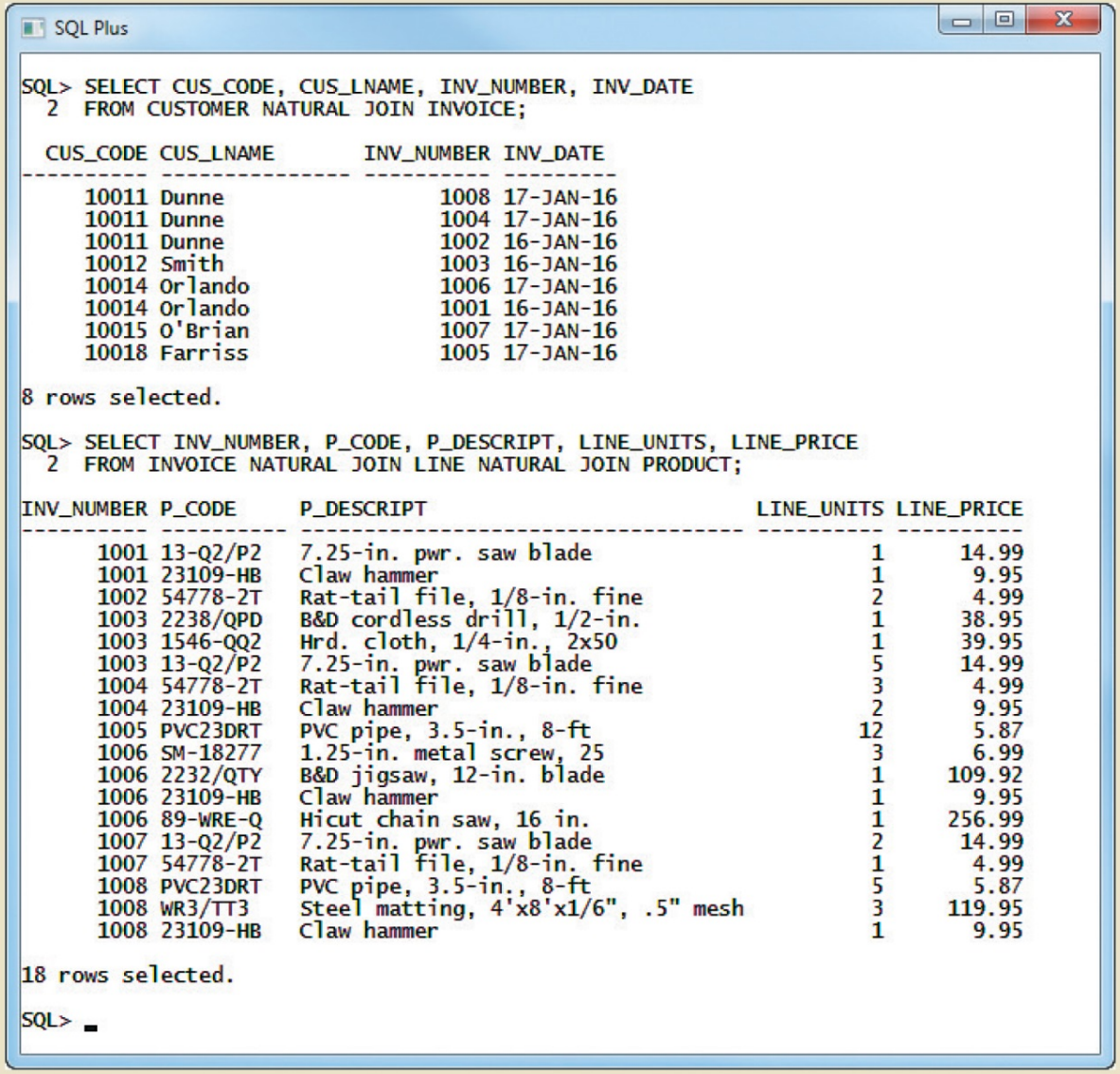


FIGURE 8.2 JOIN USING RESULTS

SQL File 3*

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

Filter: Export: Autosize:

INV_NUMBER	P_CODE	P_DESCRIPT	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

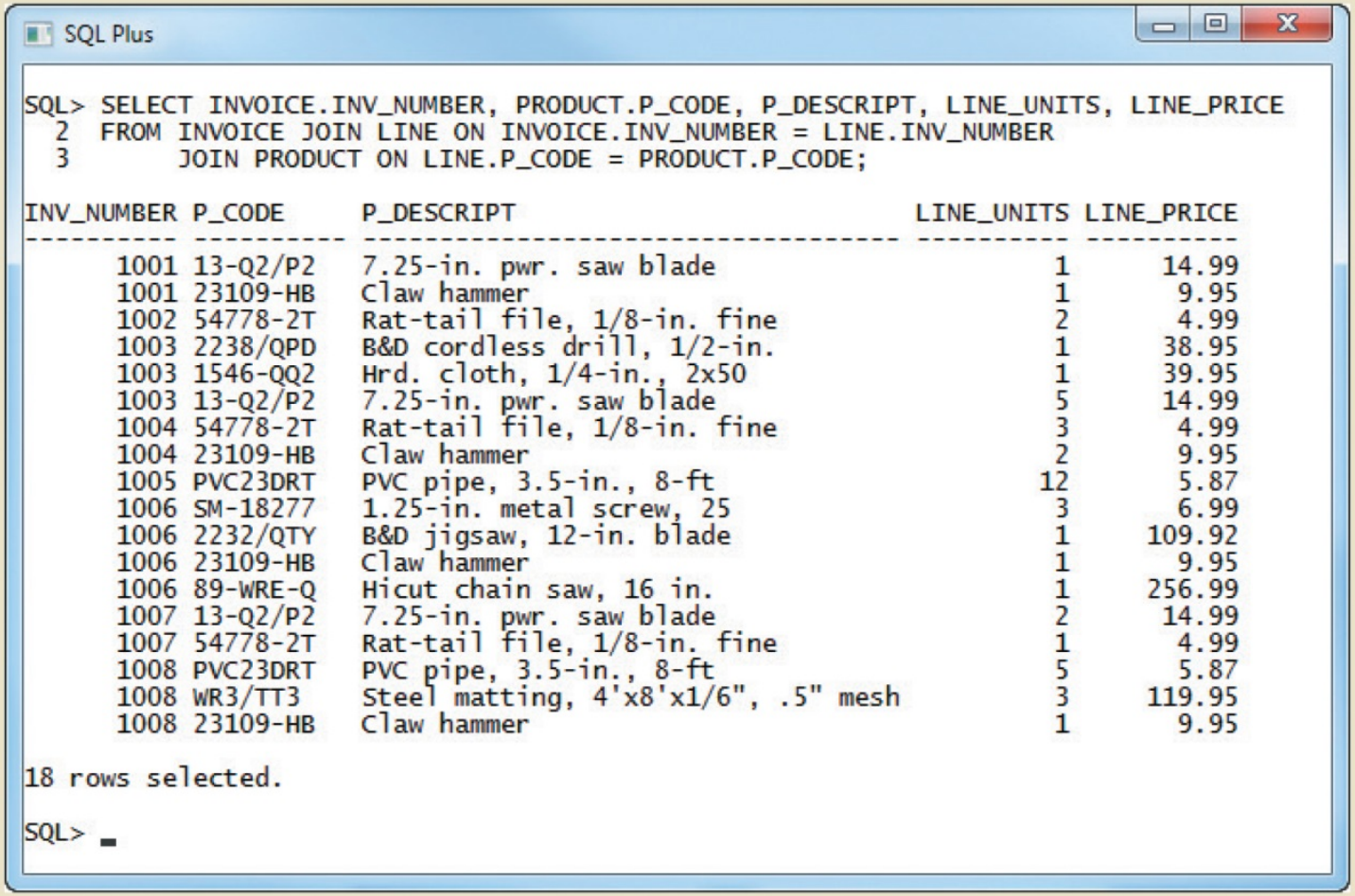


FIGURE 8.4 LEFT JOIN RESULTS

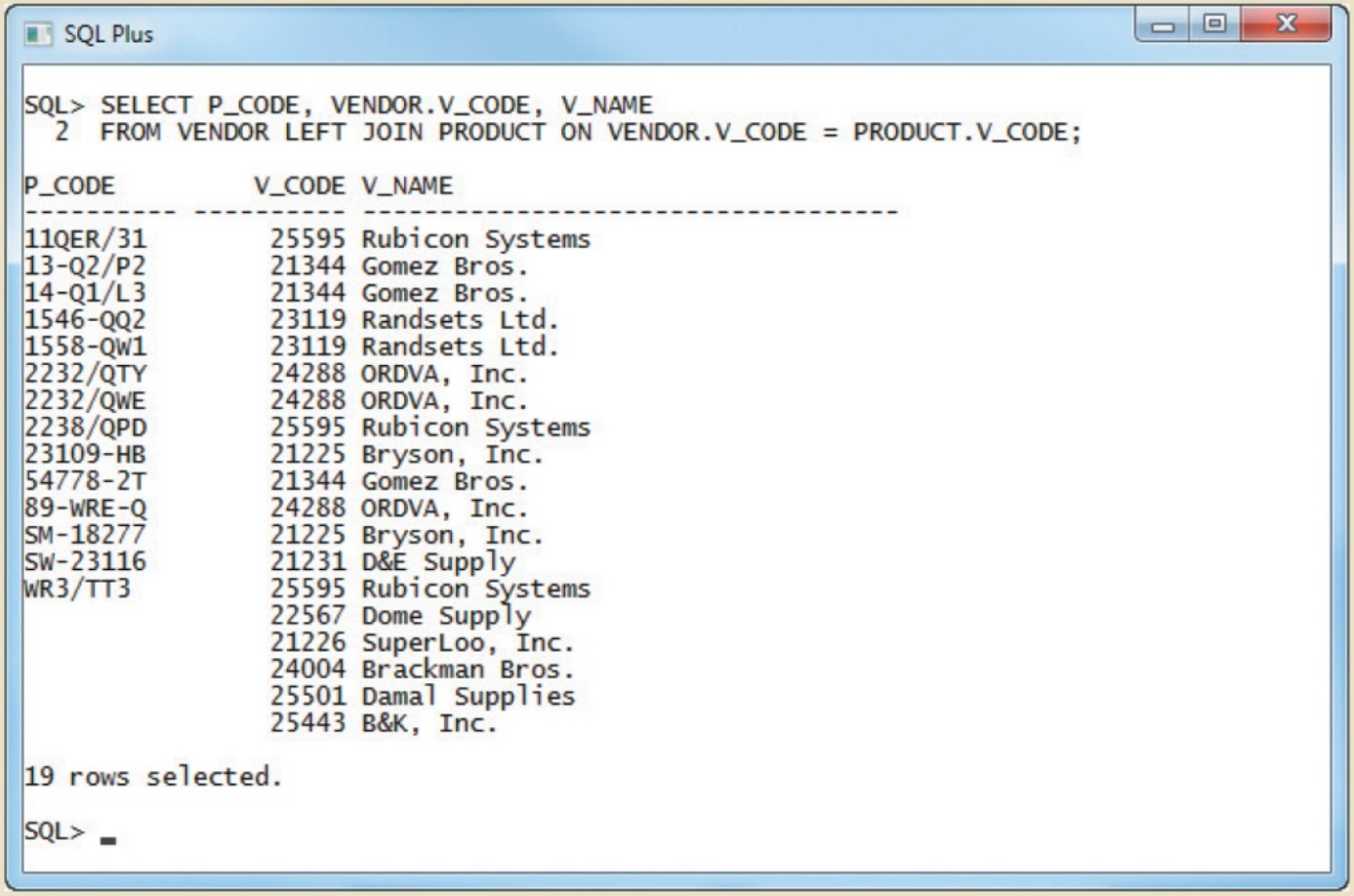


FIGURE 8.5 RIGHT JOIN RESULTS

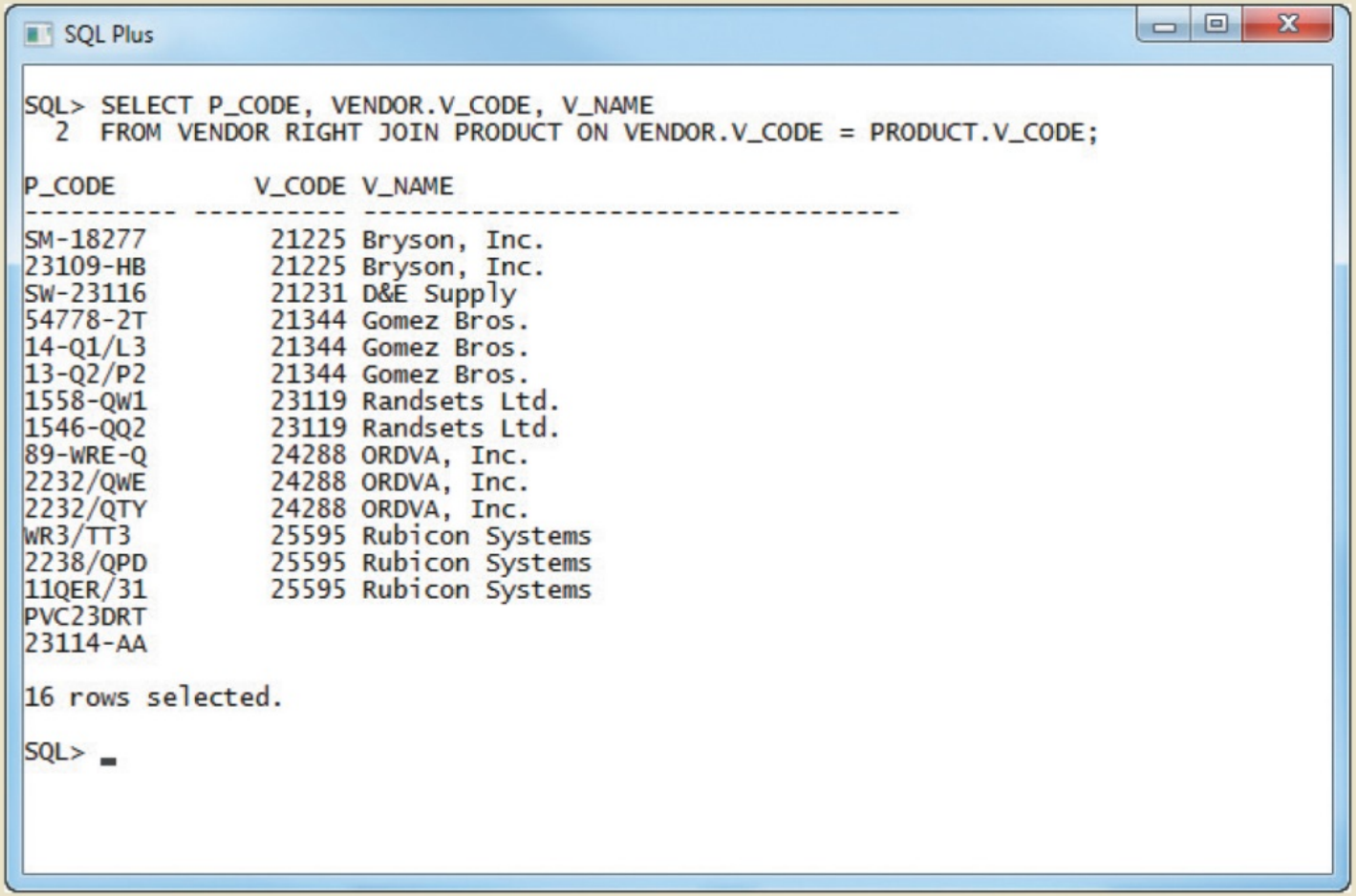
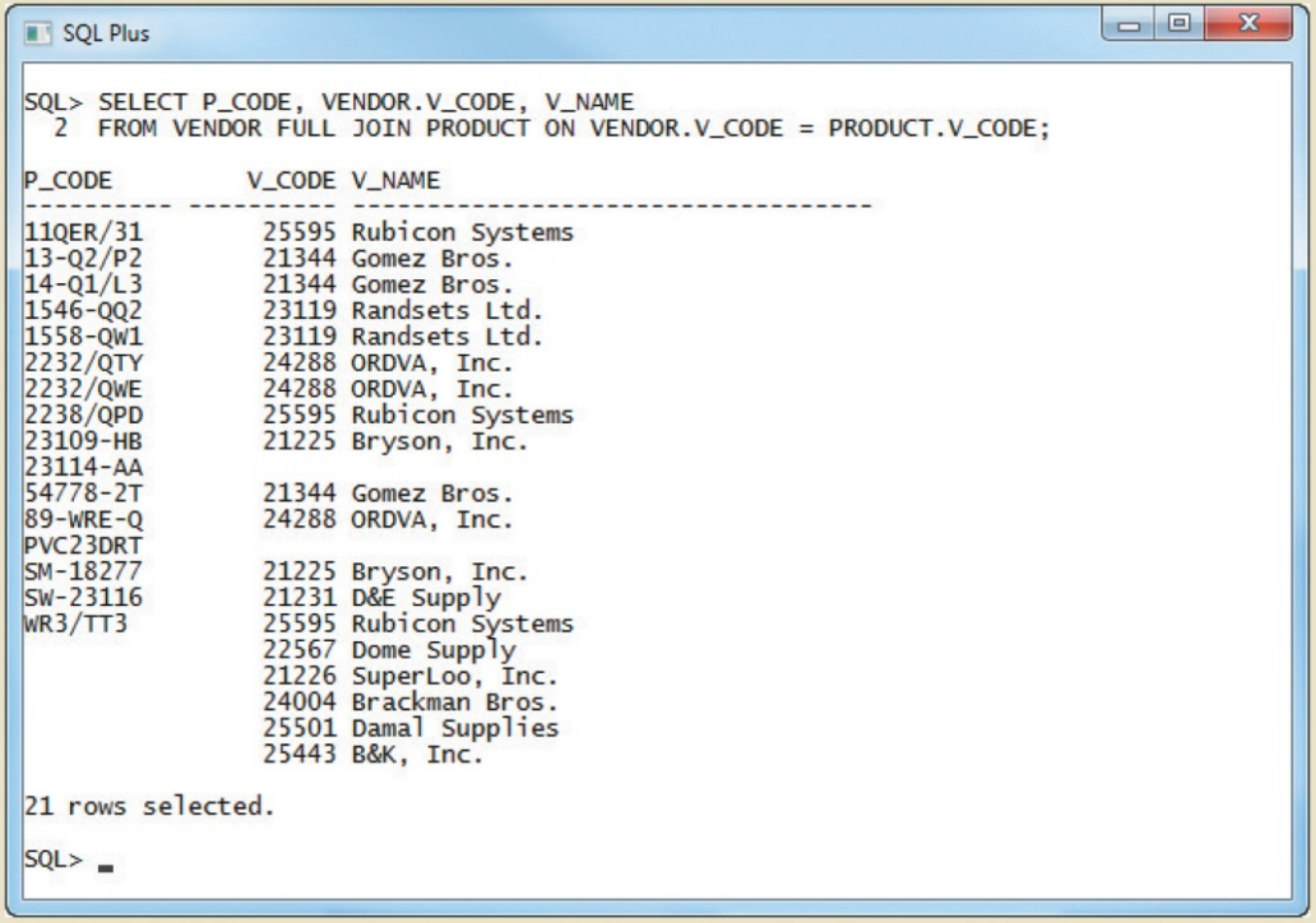


FIGURE 8.6 FULL JOIN RESULTS



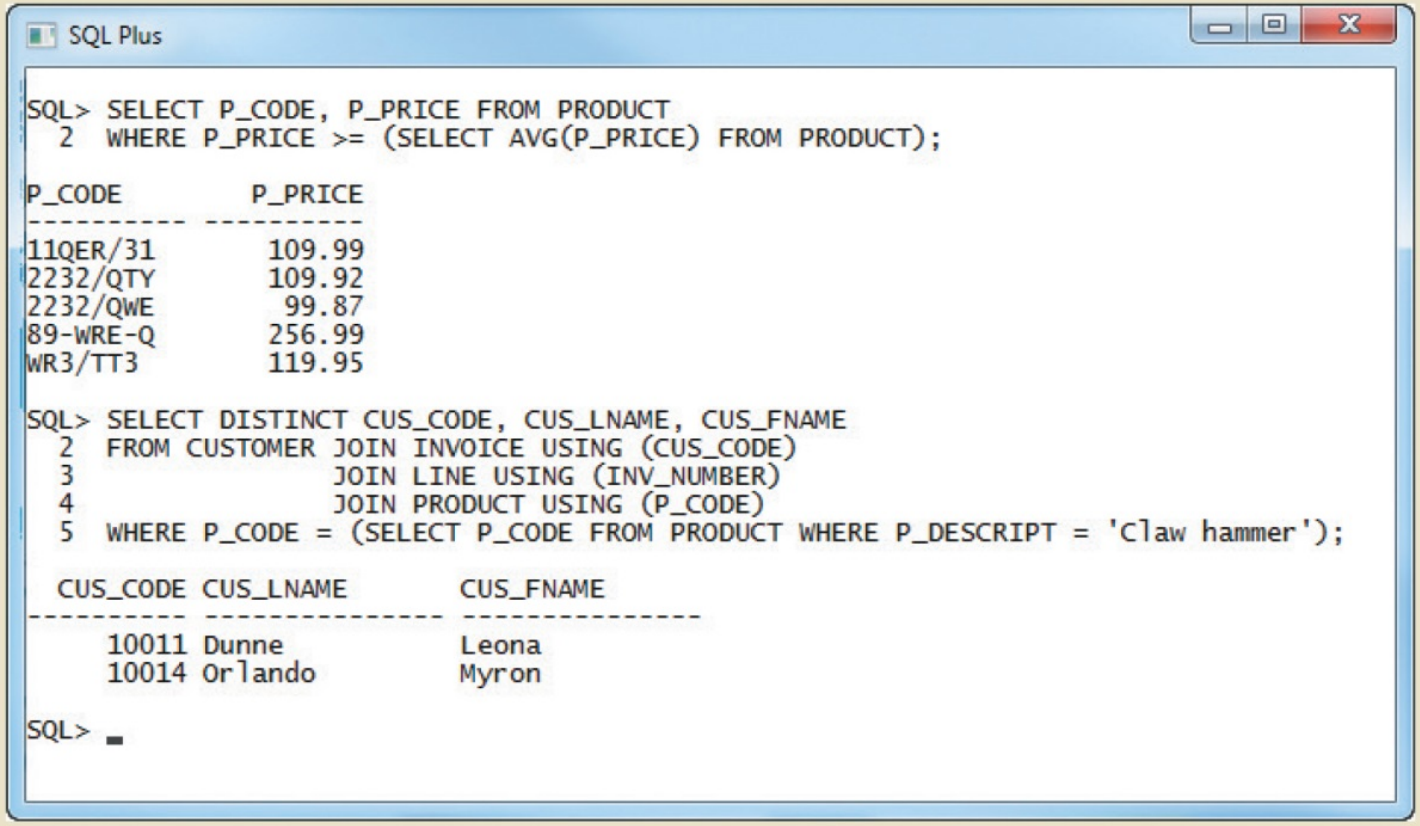
Subqueries and Correlated Queries

- 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



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

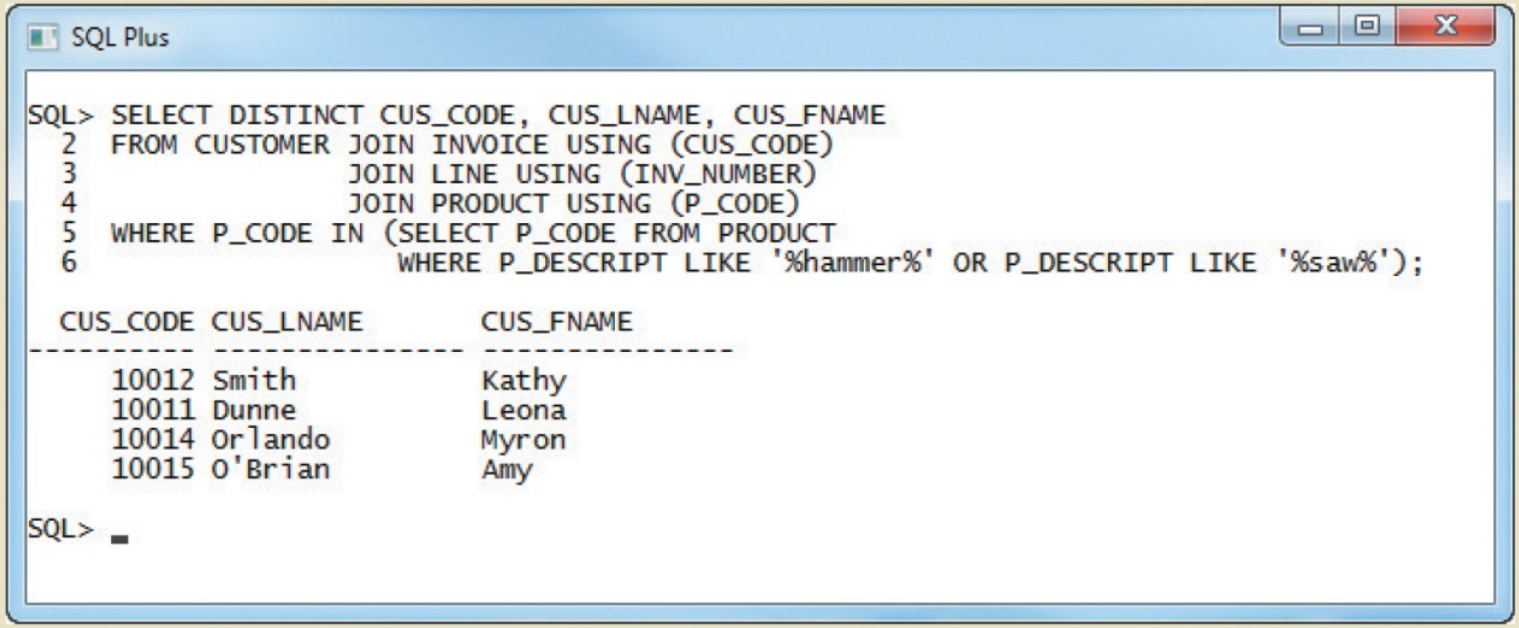
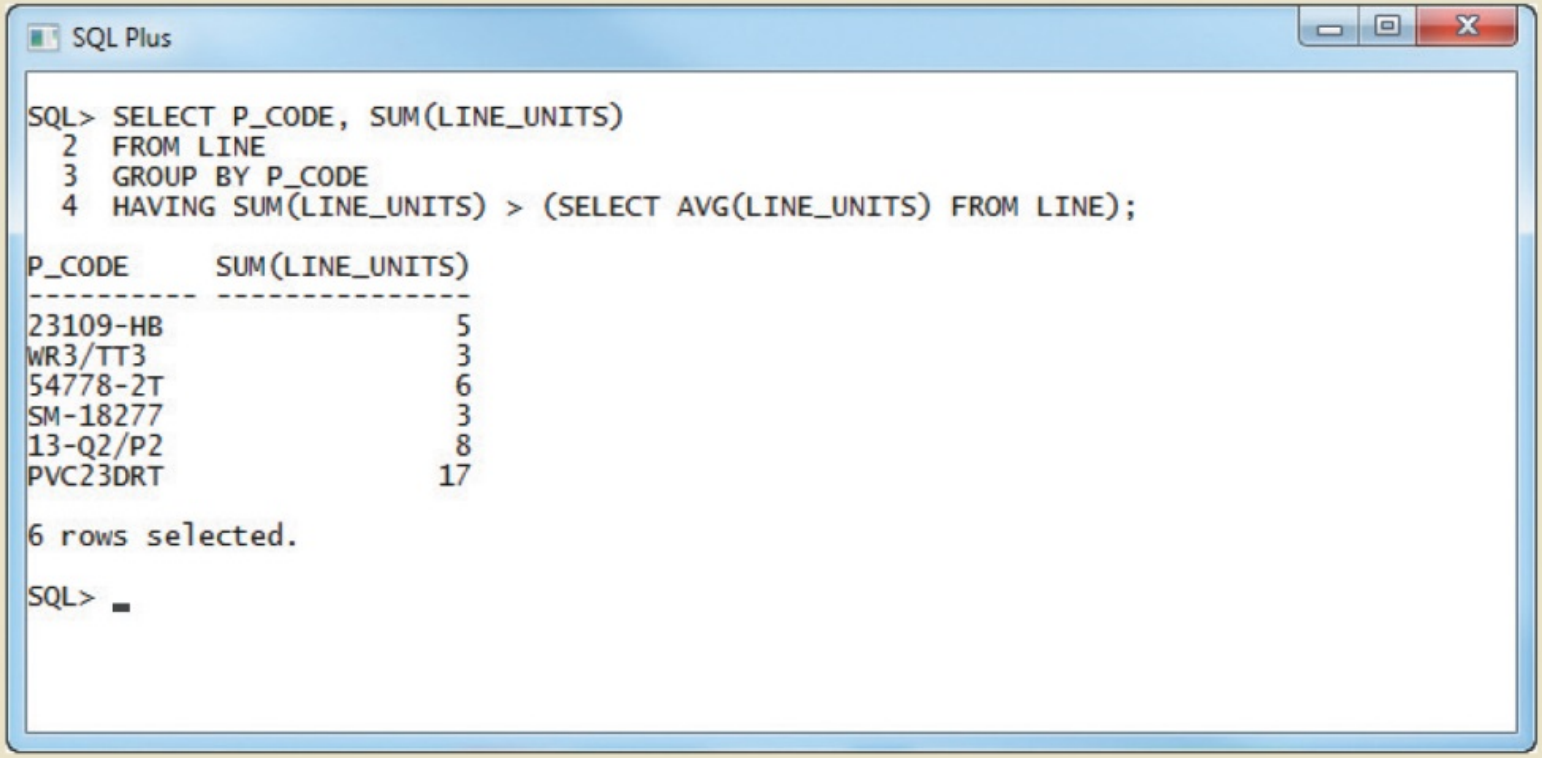


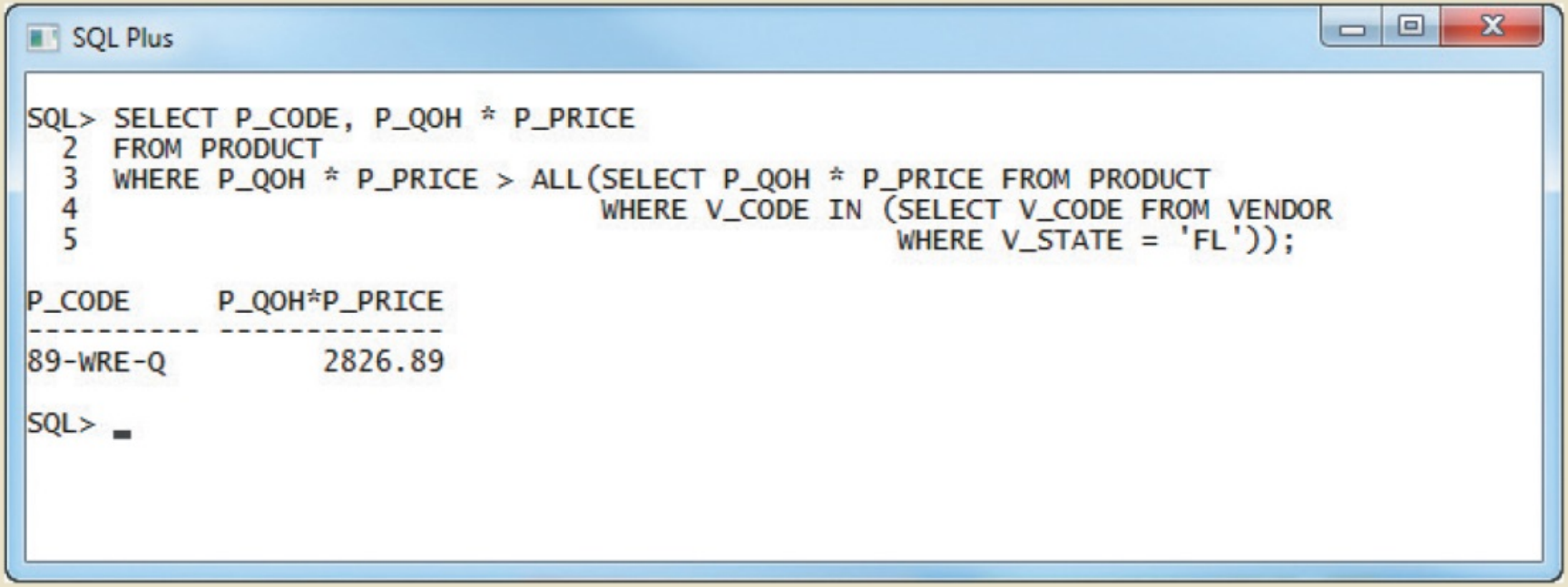
FIGURE 8.9 HAVING SUBQUERY EXAMPLE



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



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

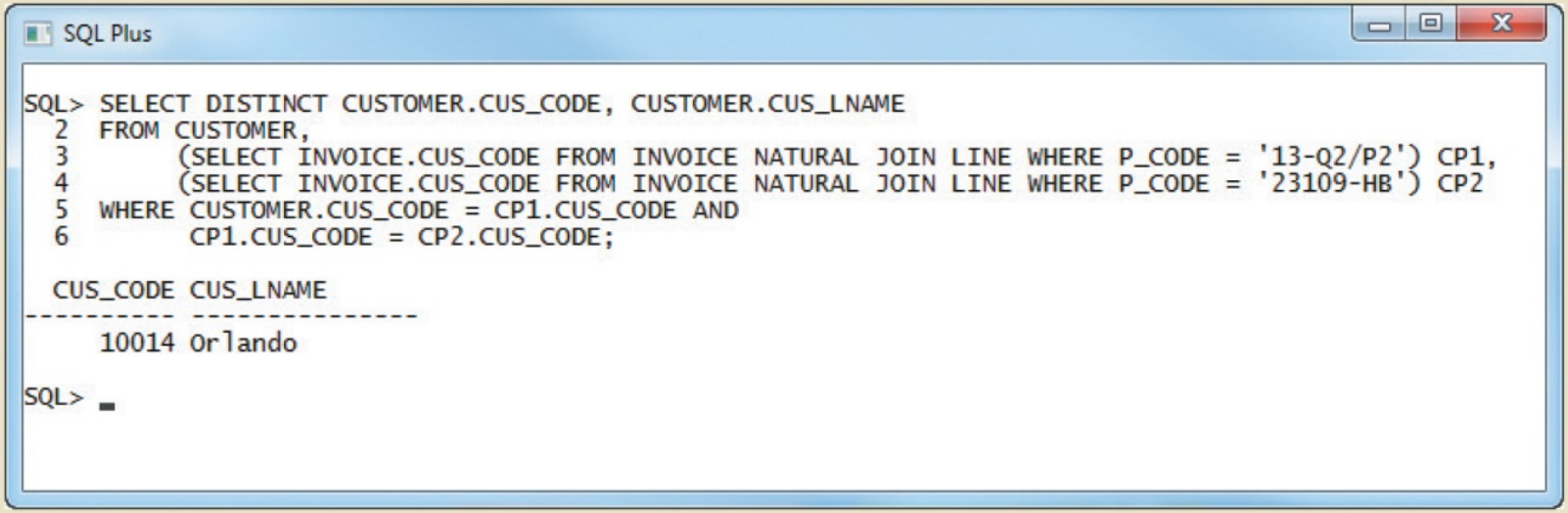


FIGURE 8.12 INLINE SUBQUERY EXAMPLE

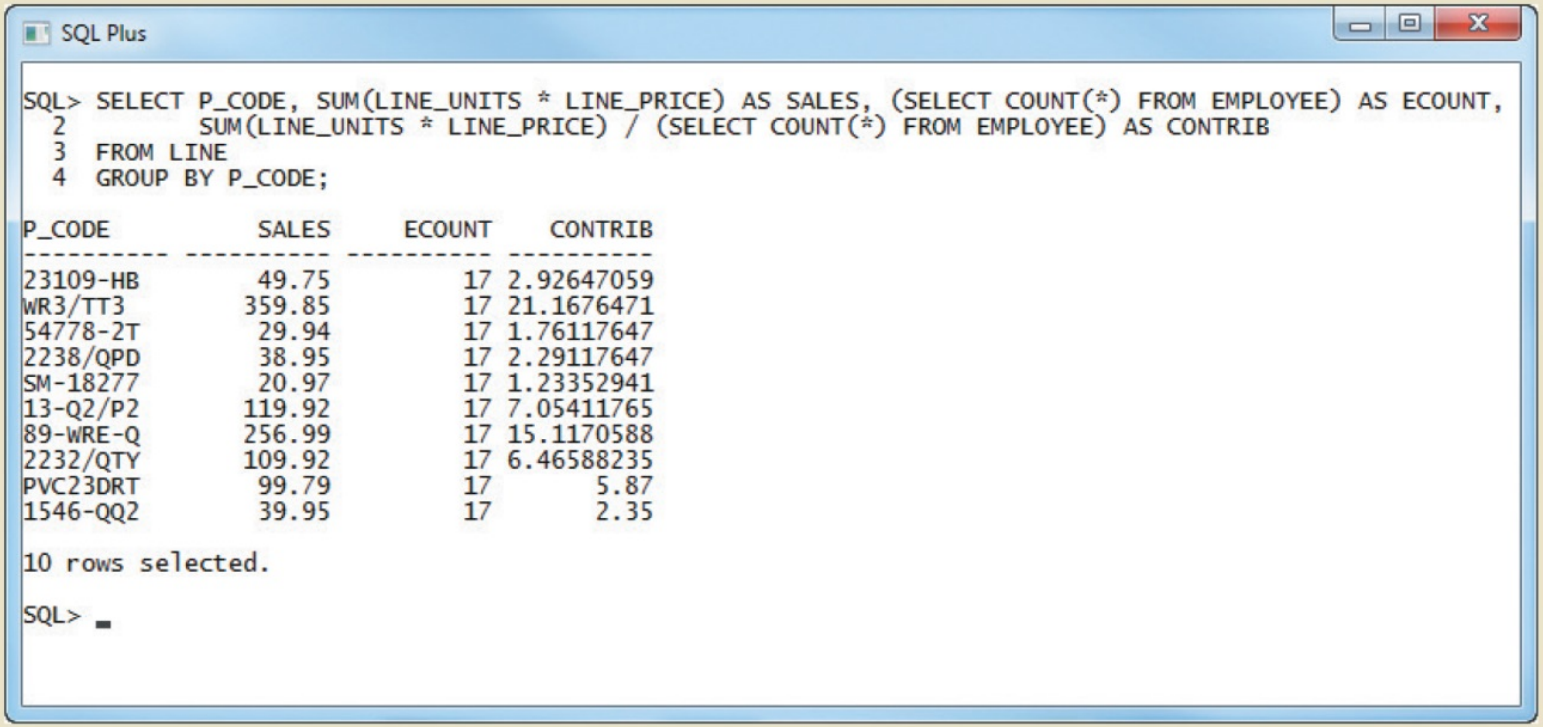
SQL File 3*

```
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;
4
```

Filter: Export: Autosize:

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	200.56874102354035
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	63.528747737407584

FIGURE 8.13 ANOTHER EXAMPLE OF AN INLINE SUBQUERY



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

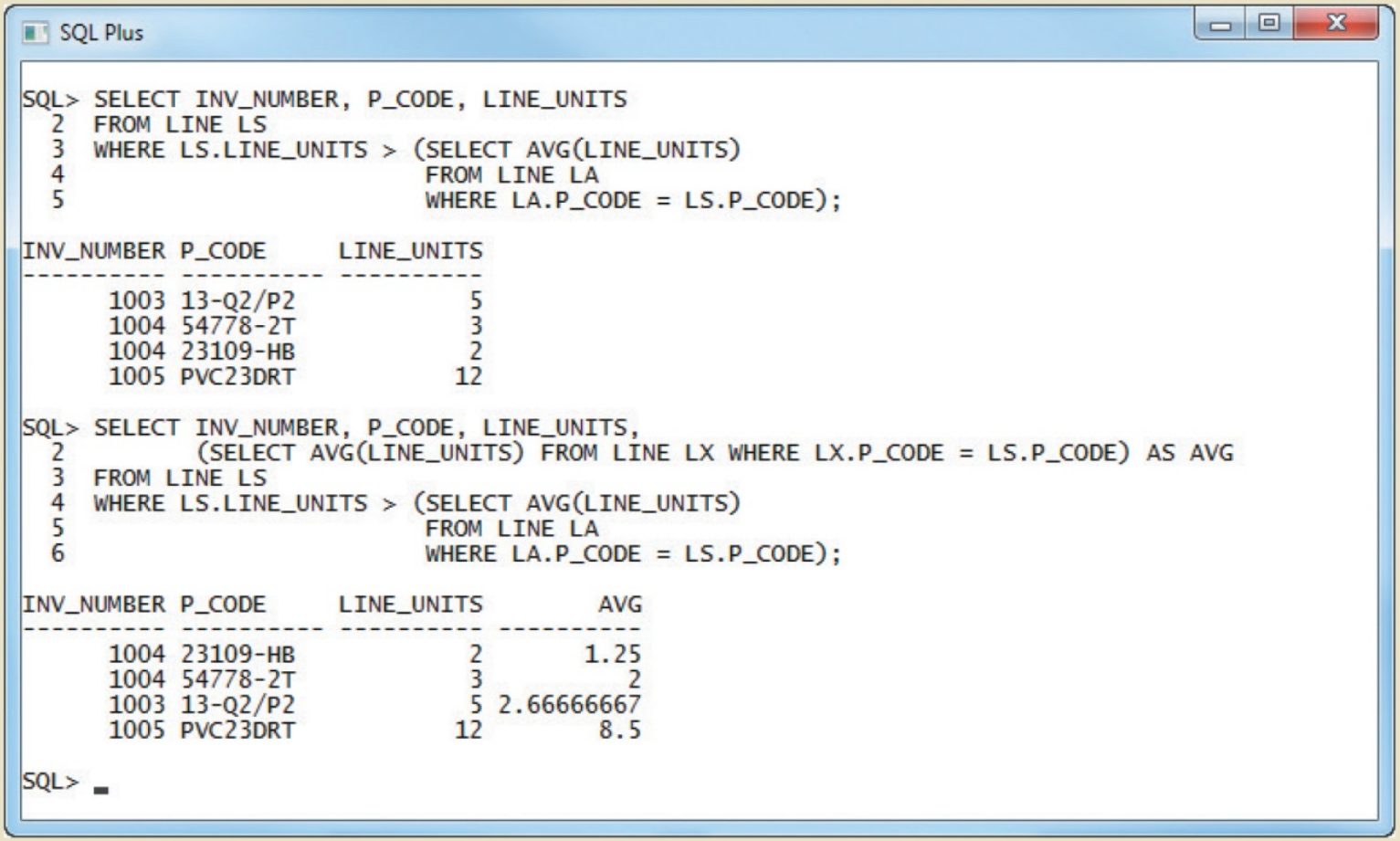
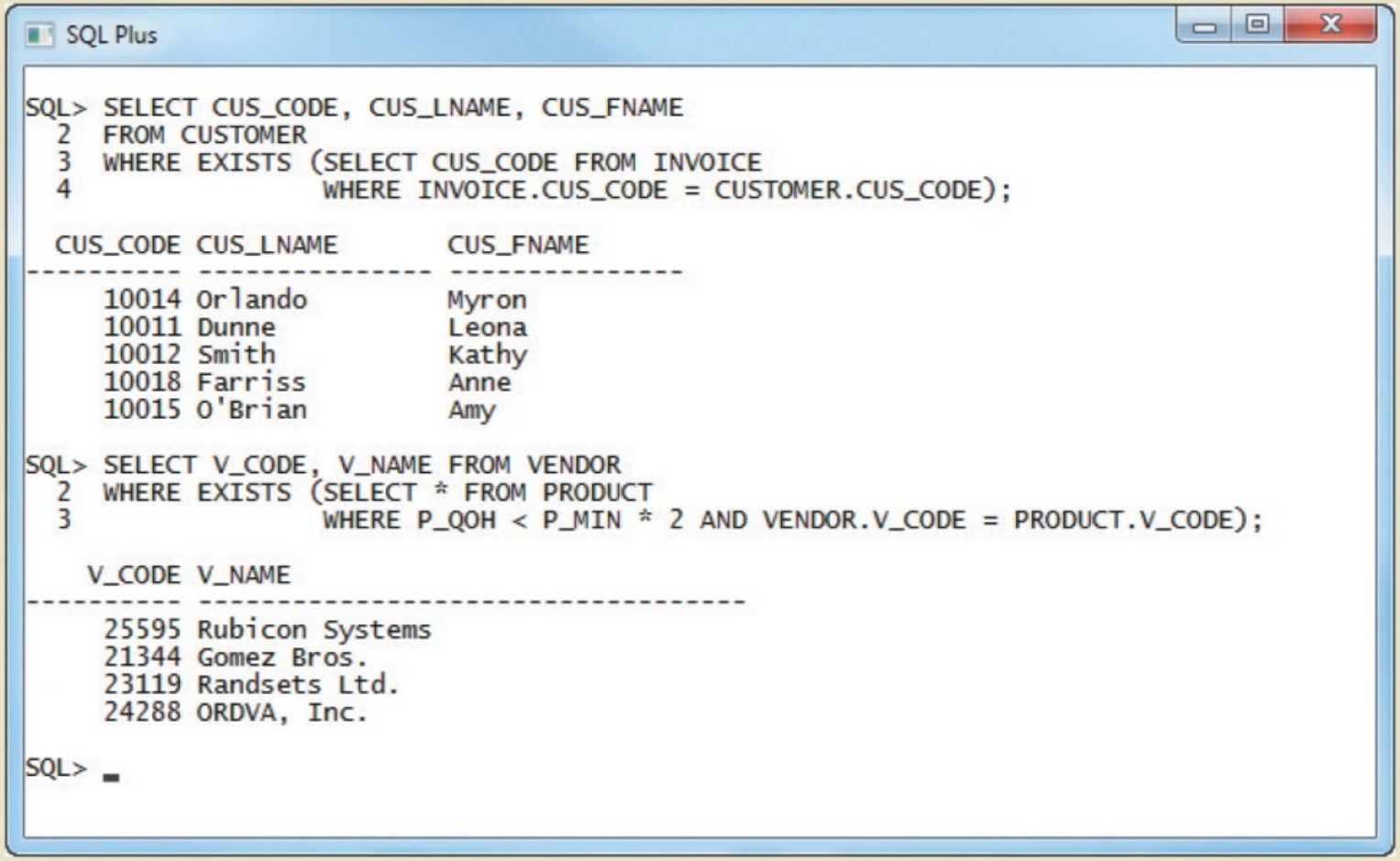


FIGURE 8.15 EXISTS CORRELATED SUBQUERY EXAMPLES



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	Olowski	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

Table name: CUSTOMER_2

CUS_CODE	CUS_LNAME	CUS_FNAME	CUS_INITIAL	CUS_AREACODE	CUS_PHONE
345	Terrell	Justine	H	615	322-9870
347	Olowski	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

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
Olowski	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	Khaleed	G	723	123-9876
Williams	George		615	290-2556

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

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	Olowski	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

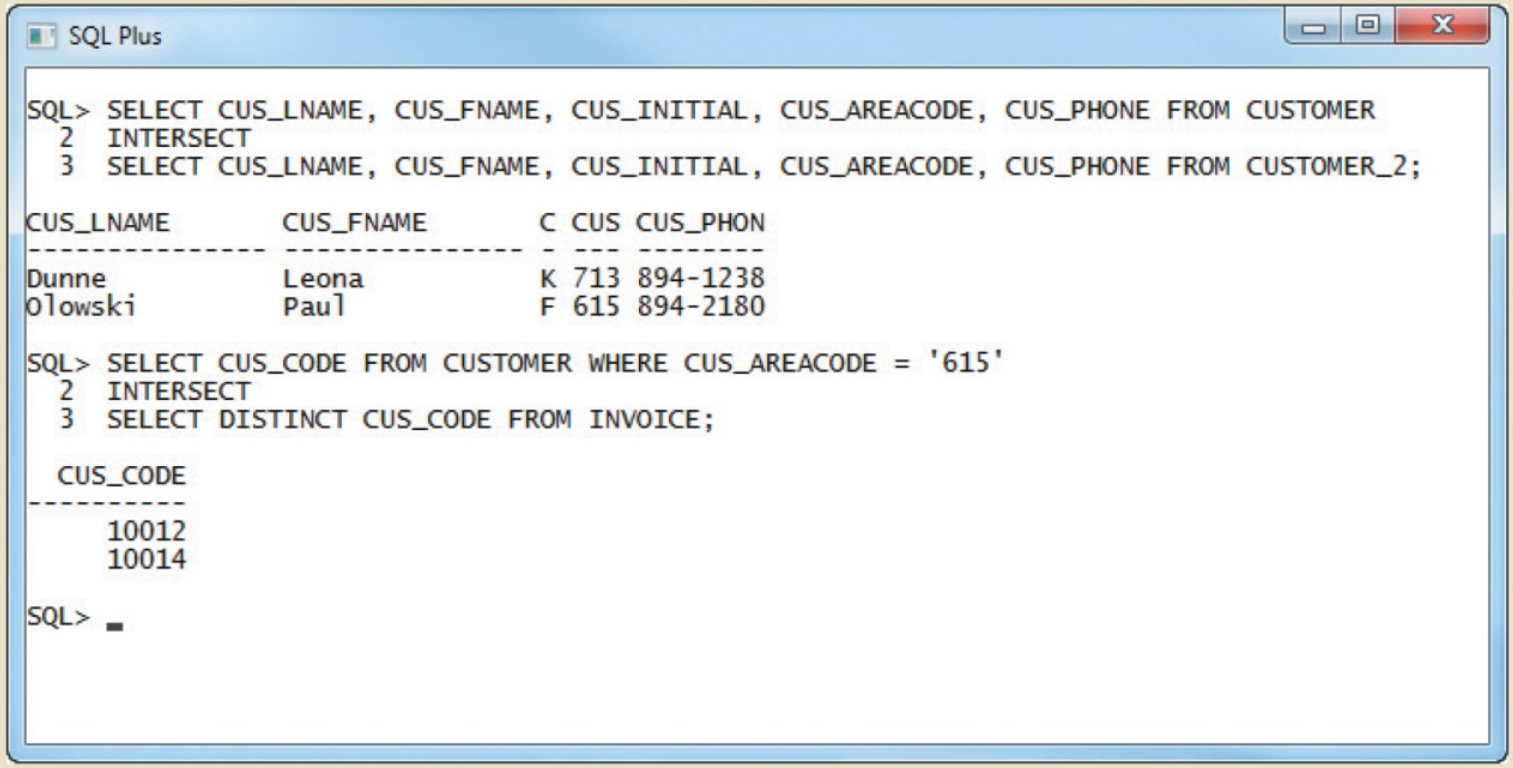
Table name: CUSTOMER_2

CUS_CODE	CUS_LNAME	CUS_FNAME	CUS_INITIAL	CUS_AREACODE	CUS_PHONE
345	Terrell	Justine	H	615	322-9870
347	Olowski	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

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

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

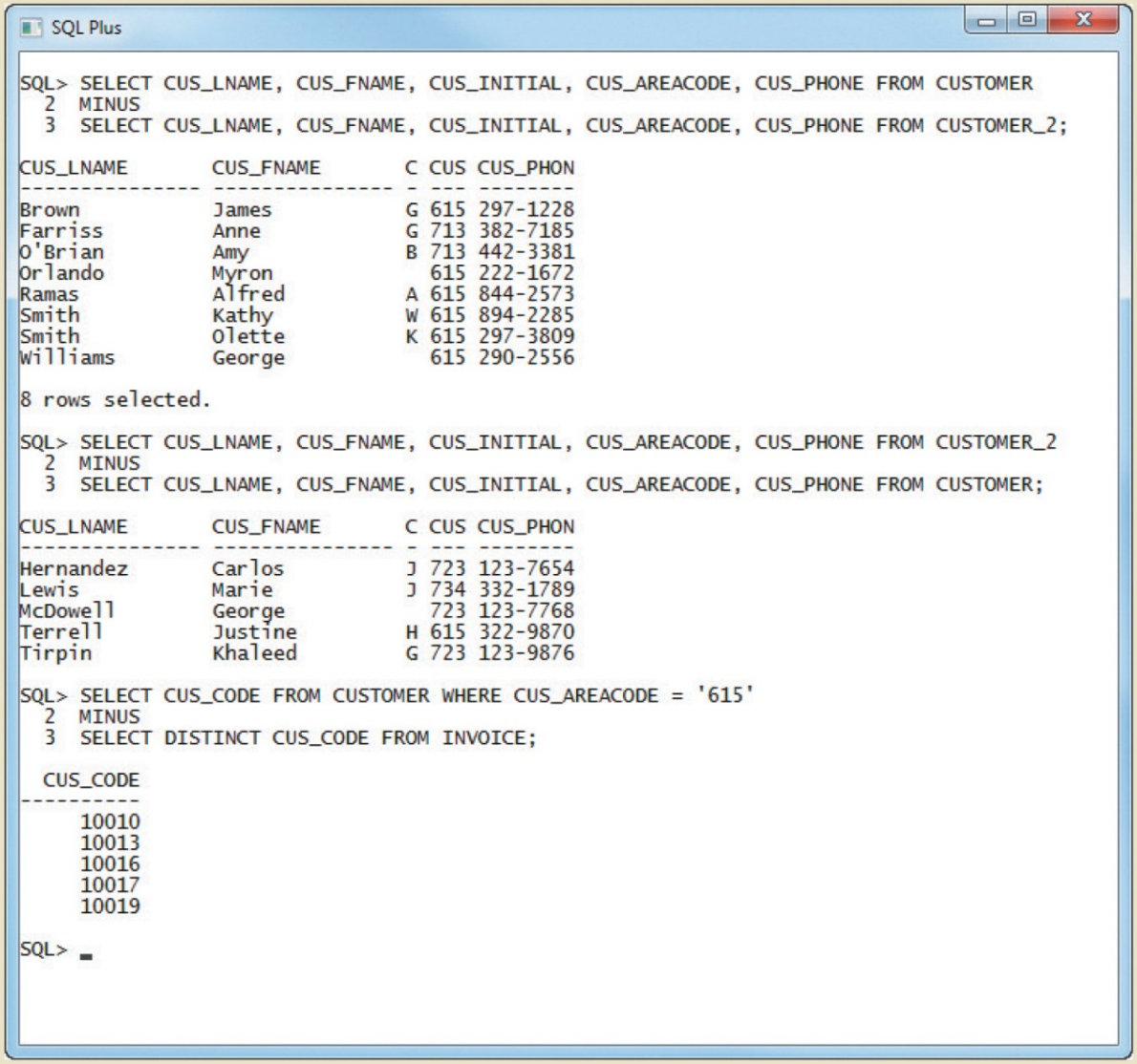
FIGURE 8.18 INTERSECT QUERY RESULTS



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



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

FIGURE 8.20 INTERSECT ALTERNATIVE

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	Olowski	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

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

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	Olowski	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

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

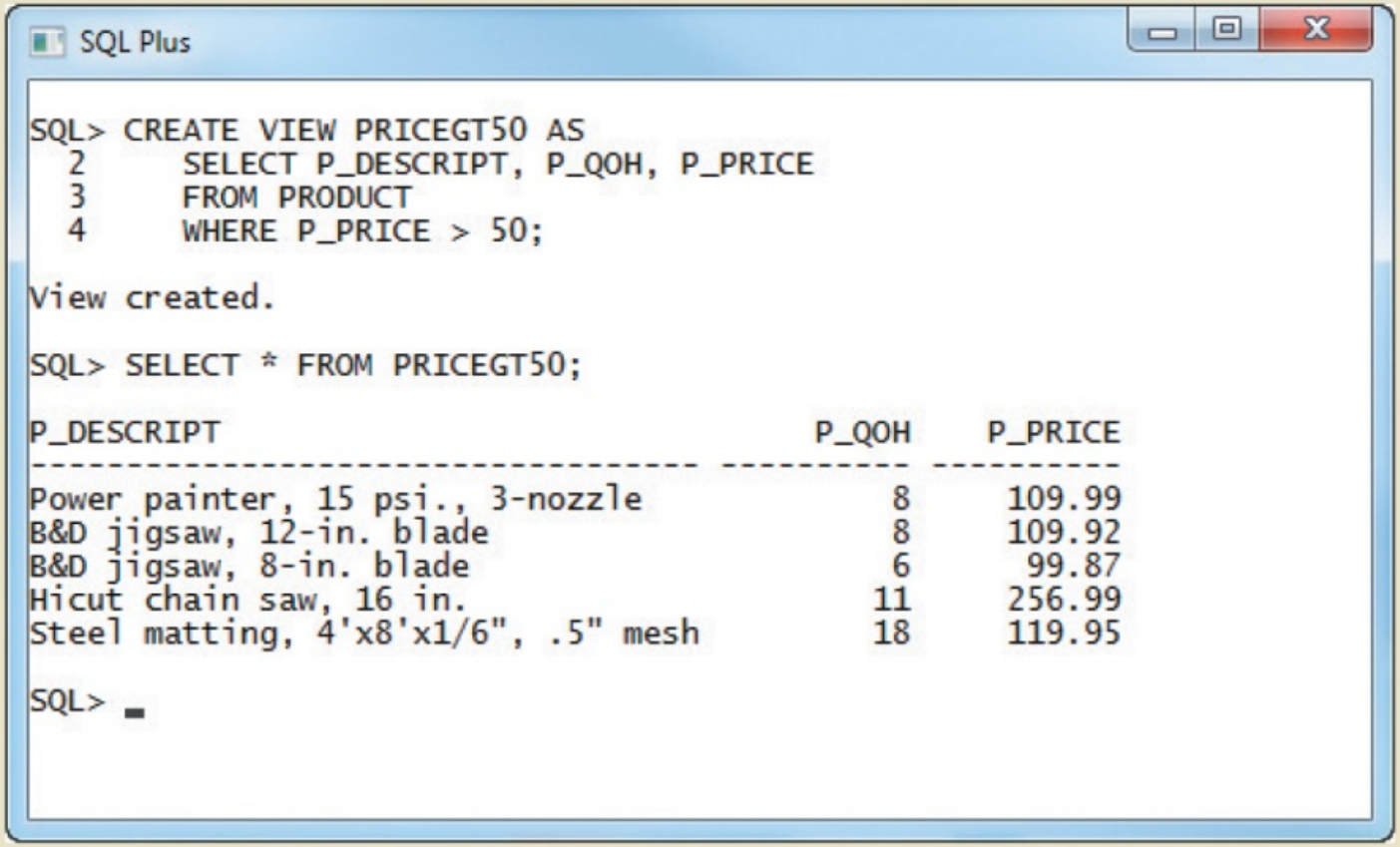


FIGURE 8.23 THE PRODMASTER AND PRODSALES TABLES

Database name: Ch08_UV

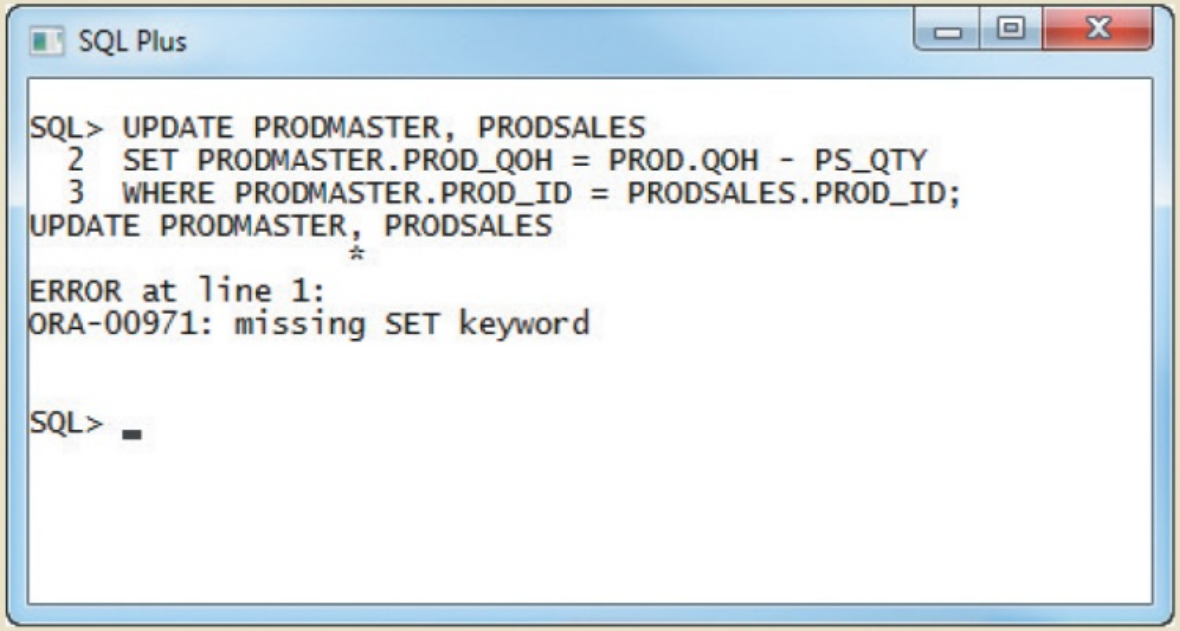
Table name: PRODMASTER

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



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

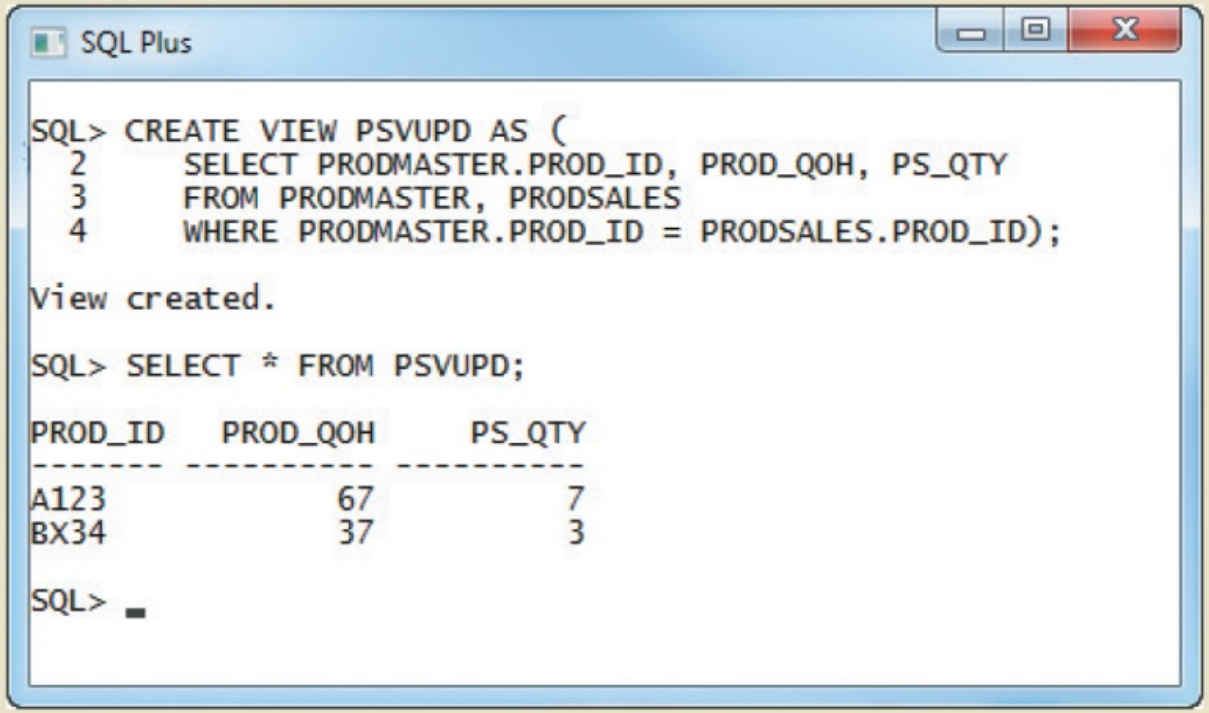
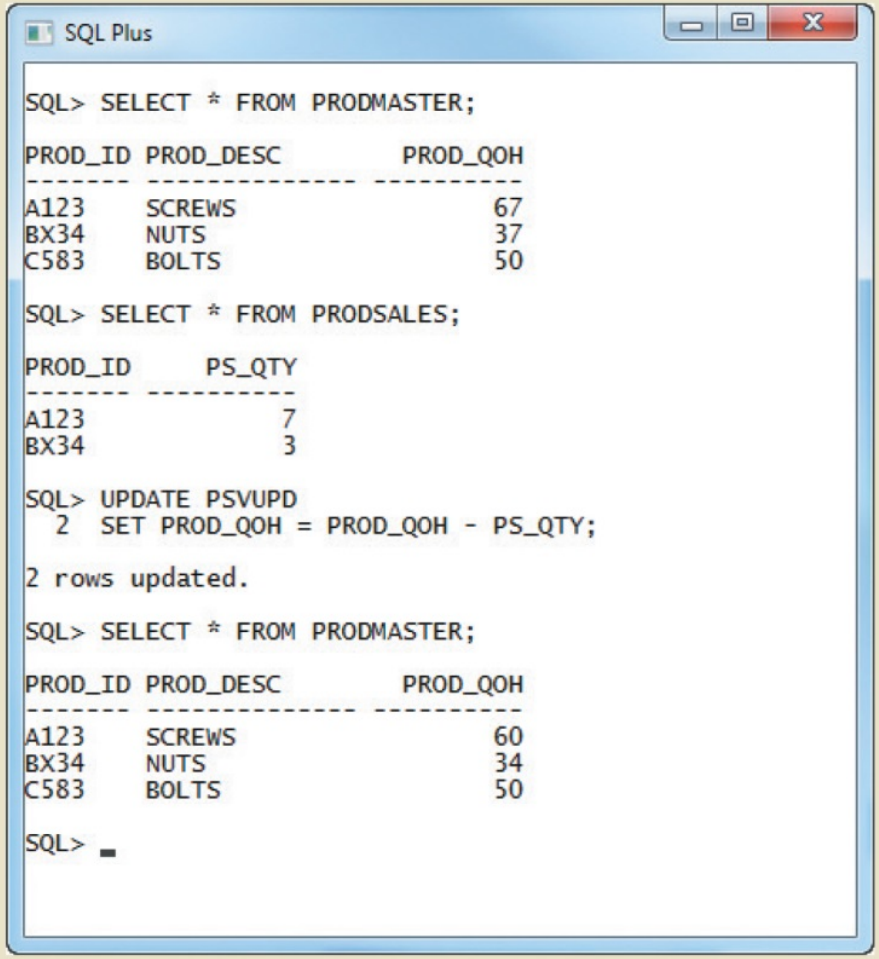


FIGURE 8.26 PRODMASTER TABLE UPDATE, USING AN UPDATABLE VIEW



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

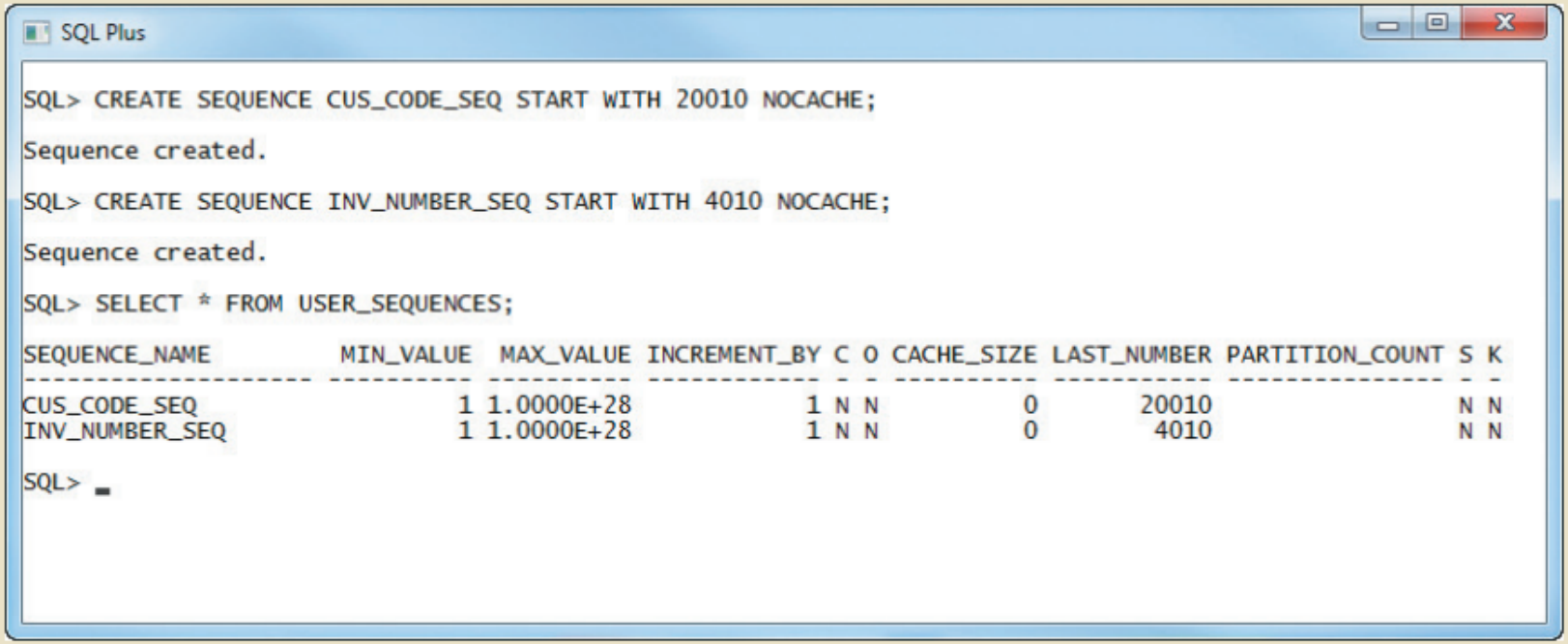
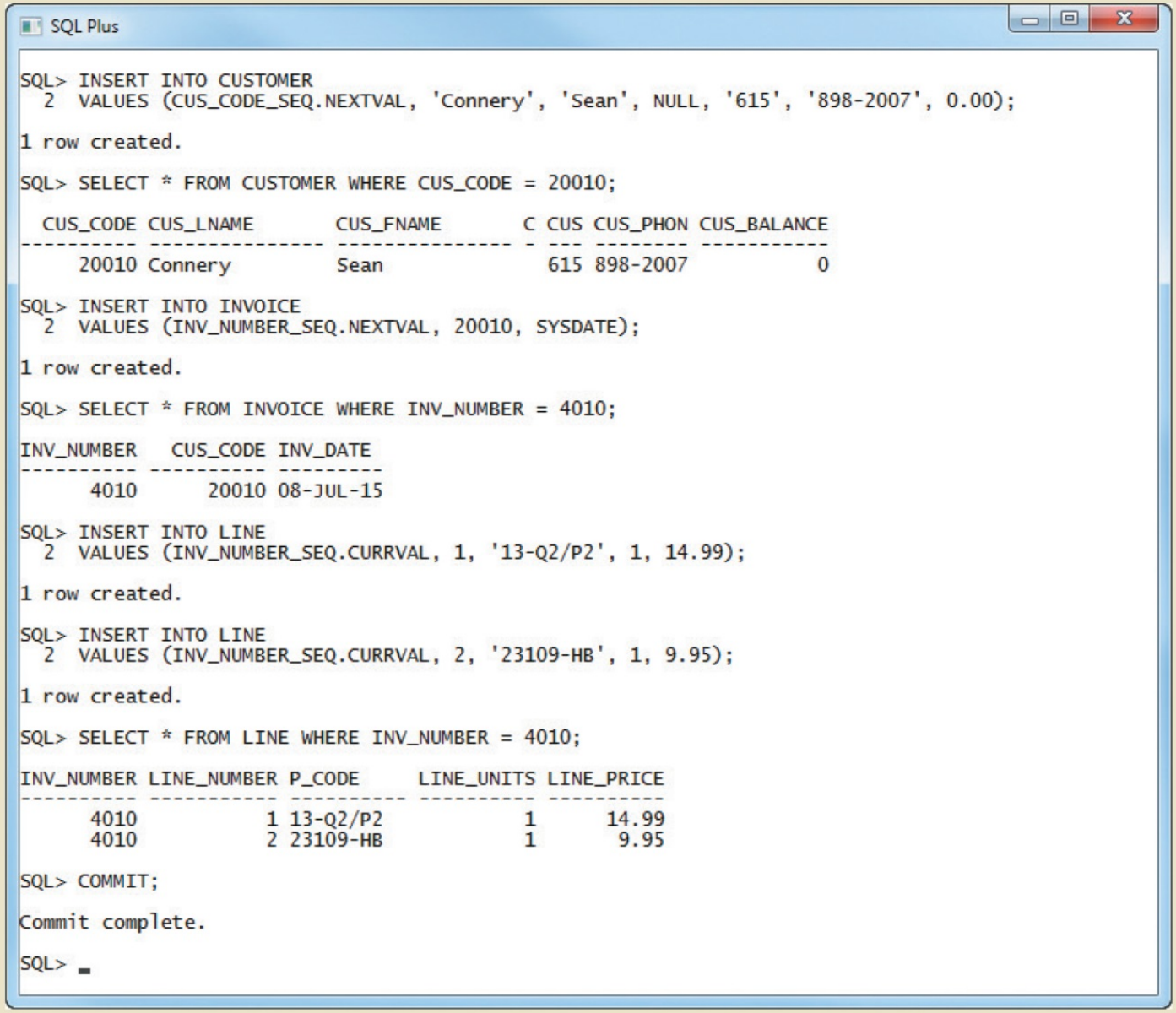


FIGURE 8.28 ORACLE SEQUENCE EXAMPLES



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	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: 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 syntax is: FETCH cursor_name INTO variable1 [, variable2, …] 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. 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.
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