

1/50    7:26:23    \*\*\*



# More SQL

# Ch.8

11e

**Database Systems**  
Design, Implementation, and Management

Coronel | Morris

Chapter 8  
Advanced SQL

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

# Objectives

## Learning Objectives

- In this chapter, the student 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

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

2

# Objectives

## Learning Objectives

- In this chapter, the student 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

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

3

# The 'JOIN' operation

Recall that we looked at examples of joining - entries from **two tables**, and entries from a **single table**. These joins were based on 'join conditions'.

It is also possible to join tables using the 'JOIN' keyword..

# JOIN conditions

Note that JOINs can be based on != (aka <>), >, <, >= and <= as well, in addition to equality. Eg. to list all students who will be getting a 'A' [uses two inequality comparisons indirectly]:

```
//  
http://www.comp.nus.edu.sg/~ooibc/courses/sql/dml\_query\_join.htm  
SELECT a.name, a.score  
FROM student_scores a, grade_class b  
WHERE b.grade = 'A' AND a.score BETWEEN  
b.low_end AND b.high_end;
```

## SQL Join Operators

- Relational join operation merges rows from two tables and returns rows with one of the following
  - Natural join - Have common values in common columns
  - Equality or inequality - Meet a given join condition
  - **Outer join:** Have common values in common columns or have no matching values
  - **Inner join:** Only rows that meet a given criterion are selected

# Ways to specify JOIN conditions

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

Cengage Learning © 2015

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

# Outer vs inner vs full ('both') JOINS

Table 8.1 - SQL Join Expression Styles

JOIN CLASSIFICATION	JOIN TYPE	SQL SYNTAX EXAMPLE	DESCRIPTION
OUTER	LEFT JOIN	<code>SELECT * FROM T1 LEFT OUTER JOIN T2 ON T1.C1=T2.C1</code>	Returns rows with matching values and includes all rows from the left table (T1) with unmatched values
	RIGHT JOIN	<code>SELECT * FROM T1 RIGHT OUTER JOIN T2 ON T1.C1=T2.C1</code>	Returns rows with matching values and includes all rows from the right table (T2) with unmatched values
	FULL JOIN	<code>SELECT * FROM T1 FULL OUTER JOIN T2 ON T1.C1=T2.C1</code>	Returns rows with matching values and includes all rows from both tables (T1 and T2) with unmatched values

Cengage Learning © 2015

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

6

# Full (left+right outer) JOIN example

For example, the following query lists the product code, vendor code, and vendor name for all products and includes all product rows (products without matching vendors) as well as all vendor rows (vendors without matching products):

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

The SQL code and its results are shown in Figure 8.12.

**FIGURE 8.12** FULL JOIN results

The screenshot shows the Oracle SQL\*Plus interface with the title bar "Oracle SQL\*Plus". The command line displays the SQL query:

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

The results are displayed in a table:

P_CODE	V_CODE	V_NAME
110C8/21	25595 Rulicon Systems	
13-IV2V2	21344 Gomez Bros.	
14-QV/L9	21204 Gomez Bros.	
1549-T02	23119 Randsets Ltd.	
1558-QM1	23119 Randsets Ltd.	
2232-QIV	24288 ORLOU, Inc.	
2232-QME	24288 ORLOU, Inc.	
2238-QPD	25595 Rulicon Systems	
23109-HB	21225 Bryceco, Inc.	
54778-2T	21344 Gomez Bros.	
89-0E-0	24288 ORLOU, Inc.	
S4-18277	21225 Bryceco, Inc.	
S4-23116	21221 D&F Supply	
VR3/T13	25595 Rulicon Systems	
	25567 Dene Supply	
	21226 Superloc, Inc.	
	24104 Brickman Bros.	
	25544 Danal Supplies	
	25443 OM, Inc.	
23114-AM		
MUC230RT		

At the bottom of the SQL\*Plus window, the message "21 rows selected." is visible.

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

# 'SELECT' subqueries

SELECT SUBQUERY EXAMPLES	EXPLANATION
INSERT INTO PRODUCT SELECT * FROM P;	Inserts all rows from Table P into the PRODUCT table. Both tables must have the same attributes. The subquery returns all rows from Table P.
UPDATE PRODUCT SET P_PRICE = (SELECT AVG(P_PRICE) FROM PRODUCT) WHERE V_CODE IN (SELECT V_CODE FROM VENDOR WHERE V_AREACODE = '615')	Updates the product price to the average product price, but only for products provided by vendors who have an area code equal to 615. The first subquery returns the average price; the second subquery returns the list of vendors with an area code equal to 615.
DELETE FROM PRODUCT WHERE V_CODE IN (SELECT V_CODE FROM VENDOR WHERE V_AREACODE = '615')	Deletes the PRODUCT table rows provided by vendors with an area code equal to 615. The subquery returns the list of vendor codes with an area code equal to 615.

Cengage Learning © 2015

## 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

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

# 'WHERE' subqueries

## 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

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

9

# WHERE subquery example

FIGURE 8.13 WHERE subquery example

```
File Edit Search Options Help
SQL> SELECT P_CODE, P_PRICE FROM PRODUCT
  2 WHERE P_PRICE >= (SELECT AVG(P_PRICE) FROM PRODUCT);
P_CODE          P_PRICE
110ER/31        109.99
2222/0TV        100.99
2232/QME        99.87
80-WRE-Q        256.00
UR3/TT3         119.95

SQL> SELECT DISTINCT CUS_CODE, CUS_LNAME, CUS_FNAME
  2 FROM CUSTOMER JOIN LINEITEM USING (CUS_CODE)
  3           JOIN LINE USING (LNU_NUMBER)
  4           JOIN PRODUCT USING (P_CODE)
  5 WHERE P_CODE IN (SELECT P_CODE FROM PRODUCT WHERE P_DESCRIFT = 'Claw hammer');
CUS_CODE CUS_LNAME      CUS_FNAME
10011  Dunne           Leona
10014  Orlando         Myron
```

# IN, HAVING subqueries

## 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

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

11

# 'IN' subqueries

Compare against a LIST of values..

FIGURE 8.14 IN subquery example

The screenshot shows the Oracle SQL\*Plus interface with a blue header bar. The title bar says 'Oracle SQL\*Plus'. The menu bar includes 'File', 'Edit', 'Search', 'Options', and 'Help'. Below the menu is a SQL command window. The command is:

```
SQL> SELECT DISTINCT CUS_CODE, CUS_LNAME, CUS_FNAME
  2  FROM CUSTOMER JOIN INVOICE USING (CUS_CODE)
  3  JOIN LINE USING (INU_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%');
```

Below the command window is a results grid. The columns are 'CUS\_CODE', 'CUS\_LNAME', and 'CUS\_FNAME'. The data is:

CUS_CODE	CUS_LNAME	CUS_FNAME
10011	Duane	Leona
10012	Smith	Kathy
10014	Orlando	Myron
10015	O'Brian	Any

At the bottom left is the SQL prompt 'SQL>'.

# 'HAVING' subqueries

As we saw earlier, this restricts the results of a GROUP BY clause. Eg. here's how to list all products sold, whose totals are greater than the average quantity sold:

FIGURE 8.15 HAVING subquery example

The screenshot shows an Oracle SQL\*Plus session. The command entered is:

```
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 following table:

P_CODE	SUM(LINE_UNITS)
13-02/P2	8
23189-HB	5
54778-2T	6
PUC23DRT	17
SM-18277	3
WRI/TT3	9

Below the table, the message "6 rows selected." is displayed. The SQL prompt "SQL>" is at the bottom left, and the bottom right corner shows a small icon.

# ALL, ANY (inequality comparisons)

Recall that 'IN' is an equality comparison against a list. To do inequality **comparison of a value against a list of values** (eg. need to be greater than ALL, need to be less than ANY..), use ALL, ANY.

## 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

# ALL, ANY

Eg. "which products do we own [in our store], whose value is more than ALL other products's values supplied by vendors in Florida?"

FIGURE 8.16 Multirow subquery operator example

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> SELECT P_CODE, P_QOH*P_PRICE
2  FROM PRODUCT
3  WHERE P_QOH*P_PRICE > ALL
4  (SELECT P_QOH*P_PRICE FROM PRODUCT
5  WHERE V_CODE IN (SELECT V_CODE FROM VENDOR WHERE V_STATE = 'FL'));

P_CODE          P_QOH*P_PRICE
-----  -----
89-WRE-Q        2820.89

```

Note that 'greater than ALL' is eqvt to 'greater than the largest of'. 'ALL' is used to select rows [plural in general] that comparison-succeed against all values in a list.

Another powerful operator is the ANY multirow operator (the near cousin of the ALL multirow operator). The ANY operator allows you to compare a single value to a list of values, selecting only the rows for which the inventory cost is greater than any value of the list or less than any value of the list. You could use the equal to ANY operator, which would be the equivalent of the IN operator.

'ANY' is used to select rows [plural in general] that comparison-succeed with any value in a list.

Note that ' $=$  ANY(list of values)' is equivalent to the 'IN' operator (which is itself equivalent to multiple  $=$  conditions joined by ORs). So the following are all equivalent, for a given value of 'M':

**(M==6) OR (M==8) OR (M==10)**

**M IN (6,8,10)**

M = ANY (6, 8, 10)

So loosely speaking, ALL is equivalent to AND, and ANY is equivalent to OR.

# 'FROM' subqueries

A SELECT query that appears in FROM, creates a \*\*virtual table\*\* against which the main query can run.

## FROM Subqueries

- FROM clause:
  - Specifies the tables from which the data will be drawn
  - Can use SELECT subquery

# FROM subquery example

All customers who bought both specified products

FIGURE 8.17 FROM subquery example

```
Oracle SQLPlus
File Edit Search Options Help
SQL> SELECT DISTINCT CUSTOMER.CUS_CODE, CUSTOMER.CUS_LNAME
2 FROM CUSTOMER
3 WHERE CUSTOMER.CUS_CODE
4 IN (SELECT INVOICE.CUS_CODE
5 FROM INVOICE NATURAL JOIN LINE WHERE P_CODE = '13-82/22') CP1,
6 (SELECT INVOICE.CUS_CODE
7 FROM INVOICE NATURAL JOIN LINE WHERE P_CODE = '23199-08') CP2
8 WHERE CUSTOMER.CUS_CODE = CP1.CUS_CODE AND
9 CP1.CUS_CODE = CP2.CUS_CODE;
CUS_CODE CUS_LNAME
-----10014 Orlando
SQL> |
```

# Attribute list subqueries

These subqueries determine what columns get output by the main query - they can be actual (existing) columns or computed columns or results of aggregate functions.

These are also known as 'column subqueries' or 'inline subqueries'.

## Attribute List Subqueries

- 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

# Attribute subquery example

**FIGURE 8.18** Inline subquery example

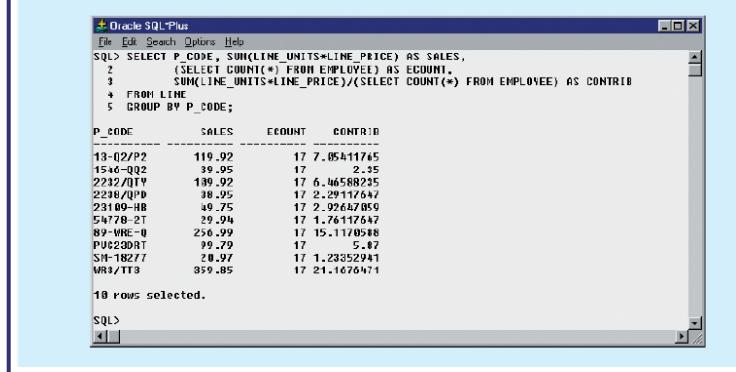
```
SQL> SELECT P_CODE, P_PRICE, (SELECT AVG(P_PRICE) FROM PRODUCT) AS AVGPRICE,
      P_PRICE - (SELECT AVG(P_PRICE) FROM PRODUCT) AS DIFF
     3  FROM PRODUCT;

P_CODE          P_PRICE    AVGPRICE      DIFF
-----          -----    -----        -----
110ER/31          189.99    56.42125   133.56875
13-0212          14.95     56.42125   -41.43125
14-0113          17.49     56.42125   -38.92725
1546-902          39.95     56.42125   -16.47125
1558-0V1          43.99     56.42125   -12.43125
2232/0IV          189.92    56.42125   133.50875
2232/0WE          99.97     56.42125   43.44675
2238/0FD          38.95     56.42125   -17.47125
23109-HD          9.95      56.42125   -46.47125
23114-FA          14.4       56.42125   -42.02125
547-00-2T          4.39      56.42125   -52.03125
0V-MW-0           250.99    56.42125   194.55125
PU2399RT          5.37      56.42125   -50.55125
SU-18277          6.99      56.42125   -49.43125
SU-23116          8.45      56.42125   -47.97125
VR3/T13          119.95    56.42125   63.52875

16 rows selected.
```

# Another attribute subquery example

FIGURE 8.19 Another example of an inline subquery



The screenshot shows the Oracle SQL\*Plus interface with a blue header bar. The main window displays a query results grid. The query itself is:

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

The results are:

P_CODE	SALES	ECOUNT	CONTRIB
13-02/P2	119.92	17	7.05411745
1546-Q02	39.95	17	2.35
2232/J0TV	139.92	17	6.46588255
2232/Q0P	38.95	17	2.2911547
231000-BB	46.75	17	2.92647803
5478-2T	29.96	17	1.7617547
89-WHE-Q	256.99	17	15.1170548
P02230R7	99.29	17	5.82
SM-18277	28.97	17	1.23352941
WR1/TT3	359.85	17	21.1676471

10 rows selected.

SQL>

# Correlated subqueries

## Correlated Subquery

- 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

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

21

In a correlated subquery, the inner [sub] query is repeatedly run, for each row of the outer query! The inner is said to be [co-]related with the outer query when it references a column in the outer query's table. This is in effect, like a double (nested) 'for' loop..

Here is the Wikipedia entry on correlated subqueries. This is the example shown there [select employees who make more than the average salary for their department]:

```
SELECT employee_number, name
  FROM employees AS Bob
 WHERE salary > (
    SELECT AVG(salary)
      FROM employees
     WHERE department = Bob.department);
```

In the above, the outer query "passes in", for each employee [each row], the employee's dept. [which the inner query refers to as Bob.department]. The inner query selects all salaries for that dept., computes the average, compares it with the passed-in employee's salary; if the test passes, the outer query selects the employee's # and name.

# Correlated subqueries [cont'd]

Until now, all subqueries you have learned execute independently. That is, each subquery in a command sequence executes in a serial fashion, one after another. The inner subquery executes first; its output is used by the outer query, which then executes until the last outer query executes (the first SQL statement in the code).

In contrast, a **correlated subquery** is a subquery that executes once for each row in the outer query. That process is similar to the typical nested loop in a programming language. For example:

```
FOR X = 1 TO 2
    FOR Y = 1 TO 3
        PRINT "X = "X, "Y = "Y
    END
END
```

1. It initiates the outer query.  
2. For each row of the outer query result set, it executes the inner query by passing the outer row to the inner query.

That process is the opposite of that of the subqueries as you have already seen. The query is called a *correlated subquery* because the inner query is *related* to the outer query by the fact that the inner query references a column of the outer subquery.

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

# Correlated subquery examples

To see the correlated subquery in action, suppose that you want to know all product sales in which the units sold value is greater than the average units sold value *for that product* (as opposed to the average for *all* products). In that case, the following procedure must be completed:

1. Compute the average units sold for a product.
2. Compare the average computed in Step 1 to the units sold in each sale row, and then select only the rows in which the number of units sold is greater.

The following correlated query completes the preceding two-step process:

```
SELECT INV_NUMBER, P_CODE, LINE_UNITS
FROM LINE LS
WHERE LS.LINE_UNITS > (SELECT AVG(LINE_UNITS)
                         FROM LINE LA
                         WHERE LA.P_CODE = LS.P_CODE);
```

```
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);

INV_NUMBER P_CODE     LINE_UNITS
----- ----- -----
1003 13-02/P2      5
1004 54778-2T      3
1004 23109-HB      2
1005 PUC23DRT      12

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);

INV_NUMBER P_CODE     LINE_UNITS      AVG
----- ----- -----
1003 13-02/P2      5  2.66666667
1004 54778-2T      3  2
1004 23109-HB      2  1.25
1005 PUC23DRT      12  8.5
```

In the top query and its result in Figure 8.14, note that the LINE table is used more than once, so you must use table aliases. In this case, the inner query computes the average units sold of the product that matches the P\_CODE of the outer query P\_CODE. That is, the inner query runs once, using the first product code found in the outer LINE table, and returns the average sale for that product. When the number of units sold in the outer LINE row is greater than the average computed, the row is added to the output. Then the inner query runs again, this time using the second product code found in the outer LINE table. The process repeats until the inner query has run for all rows in the outer LINE table. In this case, the inner query will be repeated as many times as there are rows in the outer query.

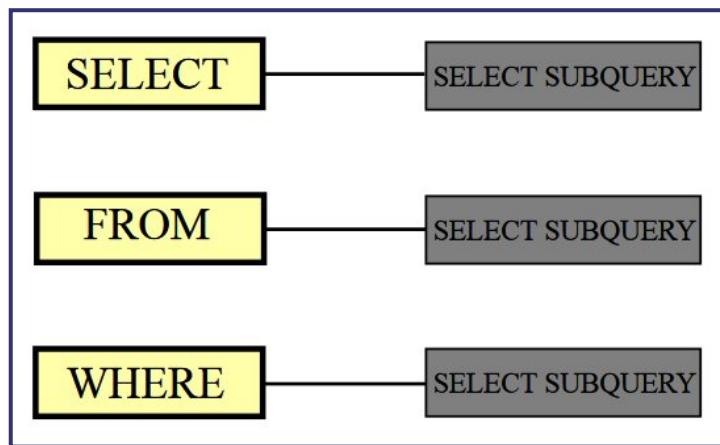
To verify the results and to provide an example of how you can combine subqueries, you can add a correlated inline subquery to the previous query. (See the second query and its results in Figure 8.14.) As you can see, the new query contains a correlated inline subquery that computes the average units sold for each product. You not only get an answer, you can also verify that the answer is correct.

**In the second query above, we have TWO correlated subqueries [that are identical], both of which need to run for every row of the main query.**

# Queries: summary

We looked at several variations of queries and subqueries (SELECT, WHERE, HAVING, IN..).

Most interestingly, a SELECT subquery can appear at the top (SELECT), middle (FROM) or bottom (WHERE) of a parent query, which provides a flexible way to express complex logic (since such subqueries can be recursively nested):



# SQL functions

## 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

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

24

# SQL functions [cont'd]

SQL Functions

- Date and time functions
- Numeric functions
- String functions
- Conversion functions

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

# UNION, INTERSECTION, DIFFERENCE

## 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

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

26

# UNION, INTERSECTION, DIFFERENCE [cont'd]

## 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

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

27

# UNION, INTERSECTION, DIFFERENCE [cont'd]

## 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

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

28

# VIEWS

## 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

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

29

# VIEW example

Creating a Virtual Table with the CREATE VIEW Command

The screenshot shows a Windows application window titled "SQL Plus". Inside, the text "Creating a Virtual Table with the CREATE VIEW Command" is displayed at the top. Below it, the following SQL code is shown:

```
SQL> CREATE VIEW PRICEGT50 AS
  2   SELECT P_DESCRPT, P_QOH, P_PRICE
  3   FROM PRODUCT
  4   WHERE P_PRICE > 50.00;

View created.

SQL> SELECT * FROM PRICEGT50;
```

After the "SELECT \* FROM PRICEGT50;" command, the results are displayed in a table:

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

At the bottom of the window, the text "Cengage Learning © 2015" is visible.

# Sequences

## 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
- Can be created and deleted any time

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

33

# Sequence creation example

Figure 8.27 - Oracle Sequence

The screenshot shows the Oracle SQL Plus interface. The title bar says "Figure 8.27 - Oracle Sequence". The main window displays the following SQL commands and their results:

```
SQL> CREATE SEQUENCE CUS_CODE_SEQ START WITH 20010 NOCACHE;
Sequence created.

SQL> CREATE SEQUENCE INU_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
INU_NUMBER_SEQ         1 1.0000E+27    1 N N      0        4010

SQL>
```

Below the command line, there is a section for inserting data into the CUSTOMER table:

```
INSERT INTO CUSTOMER
VALUES (CUS_CODE_SEQ.NEXTVAL, 'Connery', 'Sean', NULL, '615', '898-2007', 0.00);
```

At the bottom of the window, there is a copyright notice and the page number 34.

# Sequence: NEXTVAL, CURRVAL

NEXTVAL returns the current value, then does ++; CURRVAL just fetches the current value (does not ++ it).

The screenshot shows an Oracle SQL Plus session. The user is creating a sequence named CUS\_CODE\_SEQ with the next value set to 20010. They then insert a new customer record with CUS\_CODE set to the next value of the sequence. A select statement retrieves the newly inserted record. Next, they create an INU\_NUMBER\_SEQ sequence and insert a new invoice record with INU\_NUMBER set to the next value of the sequence. A select statement retrieves the newly inserted invoice record. Finally, they create a LINE sequence and insert two line items into the LINE table, both using the next value of the sequence. A select statement retrieves the newly inserted line items.

```
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 (INU_NUMBER_SEQ.NEXTVAL, 20010, SYSDATE);
1 row created.

SQL> SELECT * FROM INVOICE WHERE INU_NUMBER = 4010;
   INU_NUMBER CUS_CODE INU_DATE
-----|-----|-----|
  4010       20010 25-JUL-11

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

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

SQL> SELECT * FROM LINE WHERE INU_NUMBER = 4010;
   INU_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.
```

# Procedural Language SQL (PL/SQL)

PL/SQL involves extra (augmented) syntax that lets us do looping, branching, variable declaration and function declaration - these are of course not possible using 'plain' SQL.

PL/SQL can be used to create:

- **blocks of code** for one-time execution
- **triggers** - callbacks to invoke
- **stored procedures** - named procedures (no return values) for repeated calling
- **stored functions** - named functions (with return values) for repeated calling

## 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

# PL/SQL [cont'd]

## Procedural SQL

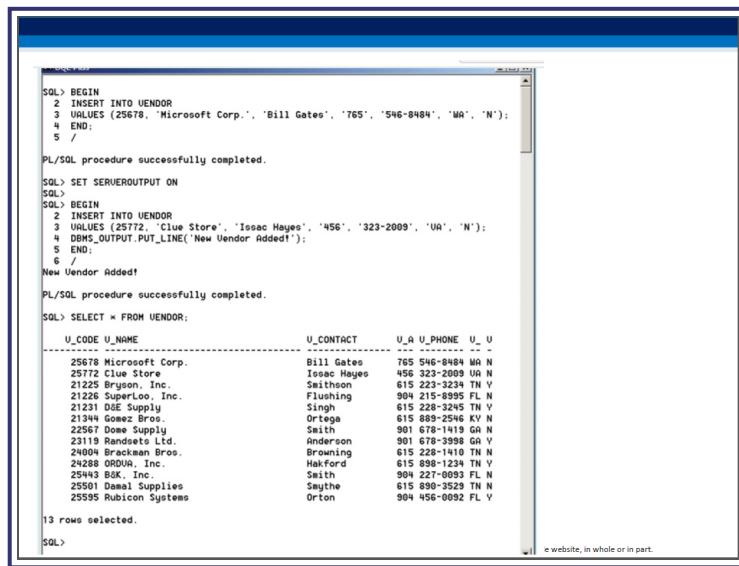
- **Procedural Language SQL (PL/SQL)**

- 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

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

37

# [Unnamed] block creation example



The screenshot shows a SQL developer interface with the following session history:

```

SQL> BEGIN
2  INSERT INTO UENDOR
3  VALUES (25678, 'Microsoft Corp.', 'Bill Gates', '765', '546-8484', 'MA', 'N');
4  END;
5 /

PL/SQL procedure successfully completed.

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

New Vendor Added!

PL/SQL procedure successfully completed.

SQL> SELECT * FROM UENDOR;

```

Below the session history is a result grid displaying vendor data:

U_CODE	U_NAME	U_CONTACT	U_A	U_PHONE	U_U
25678	Microsoft Corp.	Bill Gates	765	546-8484	MA N
25772	Clue Store	Isaac Hayes	456	323-2009	UA N
21225	Bryson, Inc.	Smithers	615	223-3234	TN V
21226	SuperLoo, Inc.	Flushing	904	215-8995	FL N
21231	D&E Supply	Singh	615	228-3245	TN V
21349	Gomez Bros.	Ortega	615	889-2594	KY N
22897	ABC Corp.	Smith	901	678-1234	TX N
23119	Randsets Ltd.	Anderson	901	678-3990	GA V
24004	Brackman Bros.	Browning	615	228-1410	TN N
24288	ORDWA, Inc.	Hakford	615	698-1234	TN V
25443	B&K, Inc.	Smith	904	227-0093	FL N
25501	Damai Supplies	Saythe	615	896-3525	TN N
25935	Rubicon Systems	Orton	904	456-0092	FL V

13 rows selected.

SQL>

A small note at the bottom right of the grid says "e website, in whole or in part."

# Triggers

## 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

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

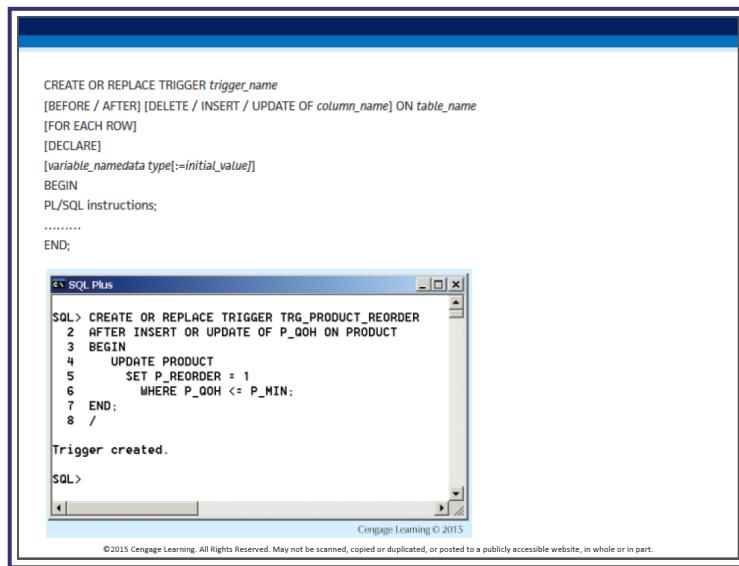
39

# Triggers [cont'd]

- *The triggering timing:* BEFORE or AFTER. This timing indicates when the trigger's PL/SQL code executes—in this case, before or after the triggering statement is completed.
- *The triggering event:* The statement that causes the trigger to execute (INSERT, UPDATE, or DELETE).
- *The triggering level:* The two types of triggers are statement-level triggers and row-level triggers.
  - A **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.
  - A **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. (In other words, if you update 10 rows, the trigger executes 10 times.)
- *The 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 ( ; ).

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

# Trigger example



The screenshot shows a Windows desktop with a blue taskbar at the bottom. On the taskbar, there are icons for Start, Task View, File Explorer, Edge browser, and others. In the center, there is a window titled "SQL\*Plus". Inside the window, the following PL/SQL code is displayed:

```
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]]
BEGIN
PL/SQL instructions;
.....
END;
```

Below this, a specific trigger definition is shown:

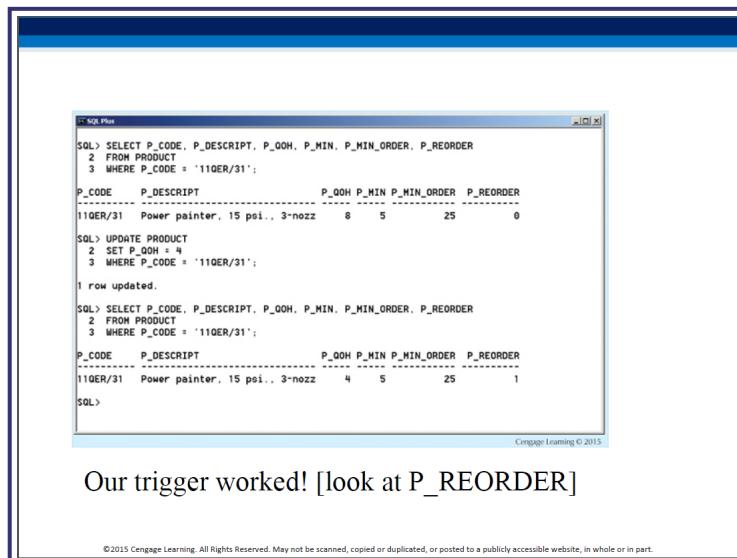
```
SQL> CREATE OR REPLACE TRIGGER TRG_PRODUCT_REORDER
2  AFTER INSERT OR UPDATE OF P_QOH ON PRODUCT
3  BEGIN
4      UPDATE PRODUCT
5          SET P_reordered = 1
6          WHERE P_QOH <= P_Min;
7  END;
8 /
```

The output of the command is:

```
Trigger created.
```

At the bottom of the SQL\*Plus window, there is a status bar with the text "Cengage Learning © 2015".

# Trigger example



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

```
SQL> SELECT P_CODE, P_DESCRIP, P_QOH, P_MIN, P_MIN_ORDER, P_REORDER
  2  FROM PRODUCT
  3  WHERE P_CODE = '11QER/31';
P_CODE      P_DESCRIP          P_QOH P_MIN P_MIN_ORDER P_REORDER
11QER/31   Power painter, 15 psi.. 3-nozz    8      5        25        0

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

1 row updated.

SQL> SELECT P_CODE, P_DESCRIP, P_QOH, P_MIN, P_MIN_ORDER, P_REORDER
  2  FROM PRODUCT
  3  WHERE P_CODE = '11QER/31';
P_CODE      P_DESCRIP          P_QOH P_MIN P_MIN_ORDER P_REORDER
11QER/31   Power painter, 15 psi.. 3-nozz    4      5        25        1
SQL>
```

Cengage Learning © 2015

Our trigger worked! [look at P\_REORDER]

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

# Triggers [cont'd]

## 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

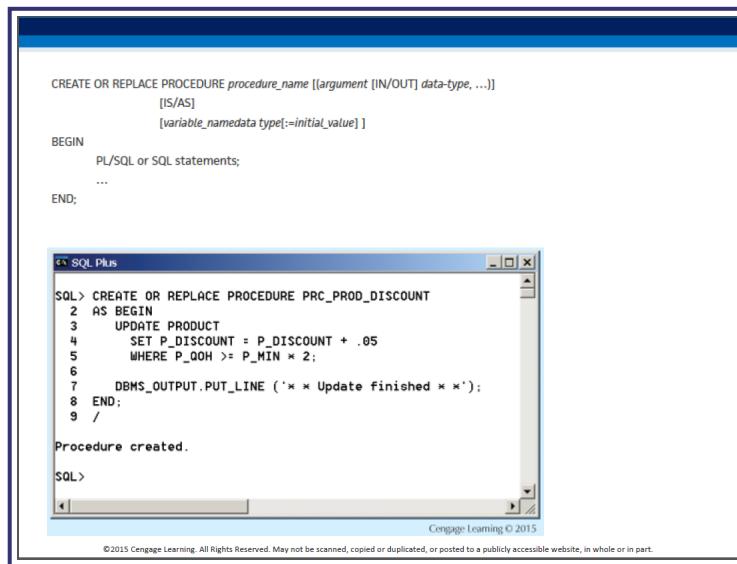
## Stored Procedures

- 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

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

44

# Stored procedure example



The screenshot shows a Windows application window titled "SQL Plus". Inside the window, there is a text area containing PL/SQL code used to create a stored procedure named PRC\_PROD\_DISCOUNT. The code includes a BEGIN block with several UPDATE statements and a DBMS\_OUTPUT.PUT\_LINE call, followed by an END; and a final /.

```
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;
```

```
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
  7      DBMS_OUTPUT.PUT_LINE ('* * Update finished * *');
  8  END;
  9 /
```

```
Procedure created.
```

```
SQL>
```

At the bottom of the window, there is a copyright notice: "©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part."

# Stored procedure example

```

SQL> SELECT P_CODE, P_DESCRIFT, P_QOH, P_MIN, P_DISCOUNT FROM PRODUCT;
P_CODE--- P_DESCRIFT----- P_QOH P_MIN P_DISCOUNT
110EZ-01 Power painter, 15 psi., 3-nozz 29 5 0.05
13-02-02 7.25-in. per. saw blade 32 15 0.05
14-Q-43 9.00-in. per. saw blade 18 12 0.00
156-Q-02 Red cloth, 1-2-in., 2x59 15 8 0.00
223-Q-01 Red cloth, 1-2-in., 3x59 0 5 0.05
223-Q-02 Red jigsaw, 8-in. blade 6 7 0.05
223-Q-03 Red jigsaw, 8-in. blade 0 5 0.05
2314-06 Sledge hammer, 12 lb. 12 10 0.05
2474-01 Rat-tail file, 1/2-in. fine 11 5 0.05
PCG30RT PVC pipe, 3.5-in., 8-ft 190 75 0.05
SM-18277 1.25-in. metal screw, 25 172 75 0.00
SM-23116 2.5-in. wd. screw, 50 237 100 0.05
WMS-T3 Steel matting, 4"x0"x1/6", .5" 18 5 0.10
16 rows selected.

SQL> UPDATE PROD SET P_DISCOUNT = 0.05 WHERE P_CODE = '110EZ-01';
** Update finished **

PL/SQL procedure successfully completed.

SQL> SELECT P_CODE, P_DESCRIFT, P_QOH, P_MIN, P_DISCOUNT FROM PRODUCT;
P_CODE--- P_DESCRIFT----- P_QOH P_MIN P_DISCOUNT
110EZ-01 Power painter, 15 psi., 3-nozz 29 5 0.05
13-02-02 7.25-in. per. saw blade 32 15 0.05
14-Q-43 9.00-in. per. saw blade 18 12 0.00
156-Q-02 Red cloth, 1-2-in., 2x59 15 8 0.00
223-Q-01 Red cloth, 1-2-in., 3x59 0 5 0.05
223-Q-02 Red jigsaw, 8-in. blade 6 7 0.05
223-Q-03 Red jigsaw, 8-in. blade 0 5 0.05
2314-06 Sledge hammer, 12 lb. 12 10 0.05
2474-01 Rat-tail file, 1/2-in. fine 11 5 0.05
PCG30RT PVC pipe, 3.5-in., 8-ft 190 75 0.05
SM-18277 1.25-in. metal screw, 25 172 75 0.00
SM-23116 2.5-in. wd. screw, 50 237 100 0.05
WMS-T3 Steel matting, 4"x0"x1/6", .5" 18 5 0.10
16 rows selected.

SQL>
```

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

# Stored functions

Reminder - these can RETURN a value.

## 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

# Stored functions - syntax

```
CREATE FUNCTION function_name (argument IN data-type, ...) RETURN data-type [IS]
BEGIN
    PL/SQL statements;
    ...
    RETURN (value or expression);
END;
```

Once such a function is defined, it can be CALLED  
inside triggers or in stored procedures..

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

# Stored functions - example

The following is an example from  
<http://www.tutorialspoint.com/plsql>.

Creating/defining a function:

```
FUNCTION findMax(x IN number, y IN number)
RETURN number
IS
    z number;
BEGIN
    IF x > y THEN
        z:= x;
    ELSE
        Z:= y;
    END IF;

    RETURN z;
END;
```

Calling/executing/running the function:

```
DECLARE
    a number;
    b number;
    c number;
BEGIN
    a:= 23;
```

```
b:= 45;  
  
c := findMax(a, b);  
dbms_output.put_line(' Maximum of (23,45):'  
' || c);  
END;  
/
```

Result:

```
Maximum of (23,45): 45
```