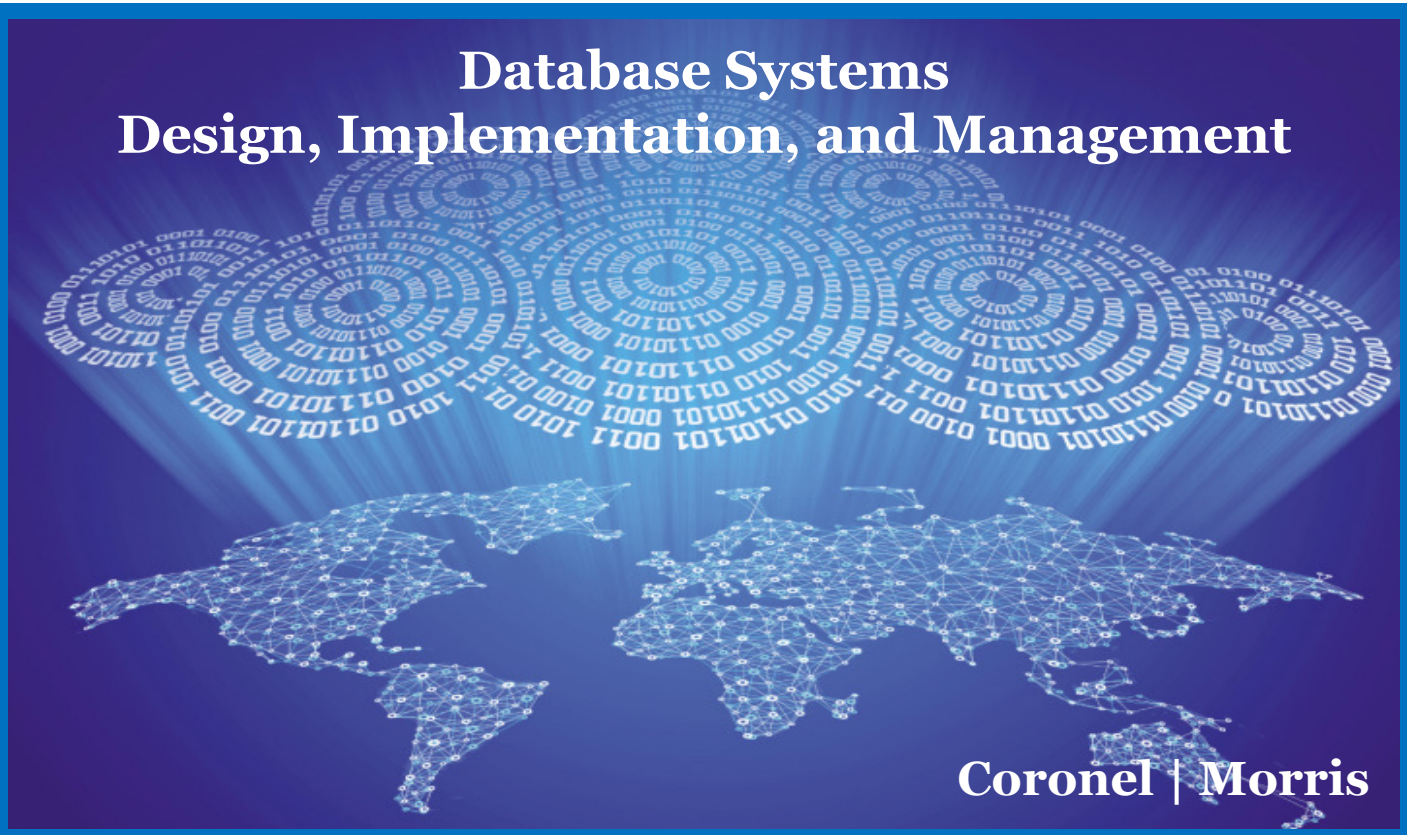
11e



Chapter 8
Advanced SQL

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

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

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

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

Table 8.1 - SQL Join Expression Styles

JOIN CLASSIFICATION	JOIN TYPE	SQL SYNTAX EXAMPLE	DESCRIPTION
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

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

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

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

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

FROM Subqueries

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

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

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

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

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

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

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

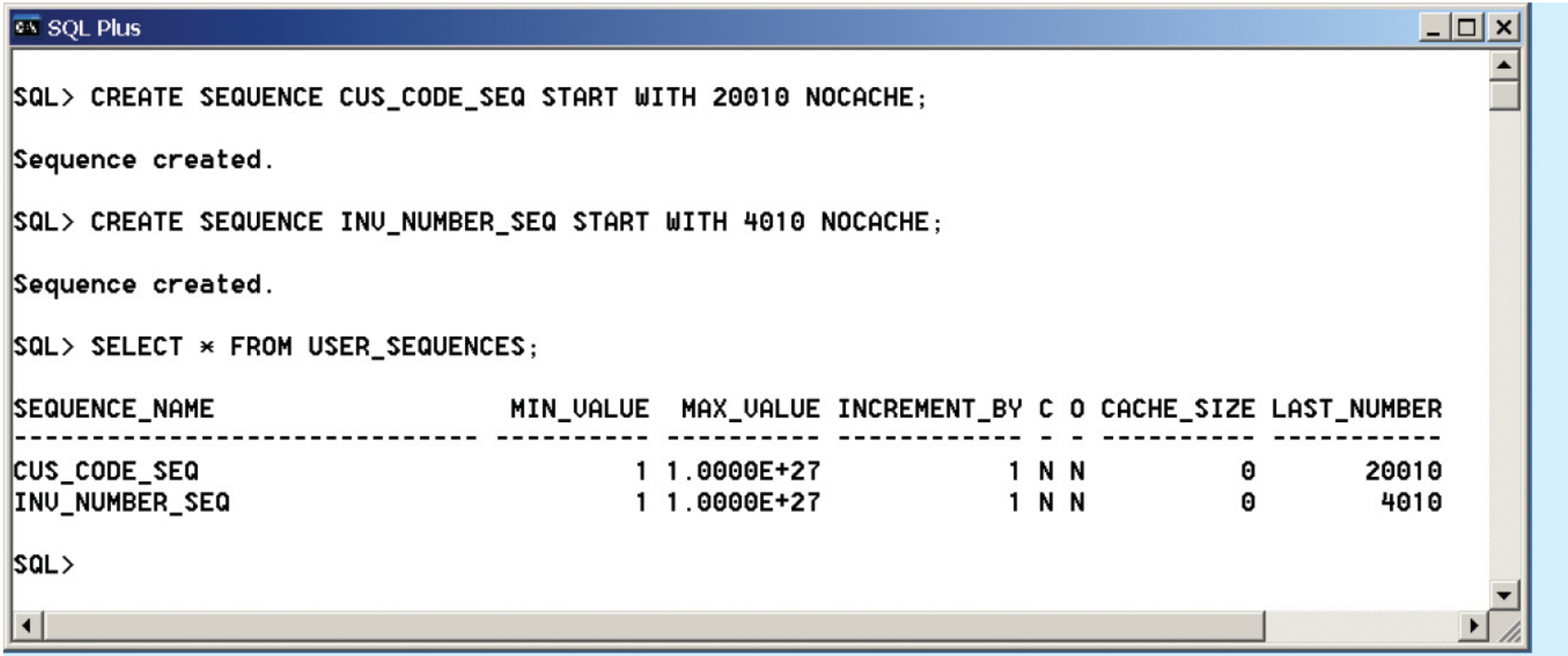
Updatable Views

- 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

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

Figure 8.27 - Oracle Sequence



Cengage Learning © 2015

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

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

Cengage Learning © 2015

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

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.

Cengage Learning © 2015

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.

Cengage Learning © 2015

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

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

Cengage Learning © 2015

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