Chapter 8

Advanced SQL

NOTE

Several points are worth emphasizing:

- We have provided the SQL scripts for the chapters. These scripts are intended to facilitate the flow of the material presented to the class. However, given the comments made by our students, the scripts should **not** replace the manual typing of the SQL commands by students. Some students learn SQL better when they have a chance to type their own commands and get the feedback provided by their errors. We recommend that the students use their lab time to practice the commands manually.
- In this chapter, the stored procedures and triggers are executed in the Oracle RDBMS. This chapter uses features that are restricted to client/server DBMSes. Such features are not commonly available in a desktop DBMS such as Microsoft Access.

Answers to Review Questions

1. What is a CROSS JOIN? Give an example of its syntax.

A CROSS JOIN is identical to the PRODUCT relational operator. The CROSS JOIN is also known as the Cartesian product of two tables. For example, if you have two tables, AGENT, with 10 rows and CUSTOMER, with 21 rows, the CROSS JOIN resulting set will have 210 rows and will include all of the columns from both tables. Syntax examples are:

```
SELECT * FROM CUSTOMER CROSS JOIN AGENT;
```

or

```
SELECT * FROM CUSTOMER, AGENT
```

If you do not specify a join condition when joining tables, the result will be a CROSS Join or PRODUCT operation.

2. What three join types are included in the OUTER JOIN classification?

An OUTER JOIN is a type of JOIN operation that yields all rows with matching values in the join columns as well as all unmatched rows. (Unmatched rows are those without matching values in the join columns). The SQL standard prescribes three different types of join operations:

LEFT [OUTER] JOIN
RIGHT [OUTER] JOIN
FULL [OUTER] JOIN.

The LEFT [OUTER] JOIN will yield all rows with matching values in the join columns, plus all of the unmatched rows from the *left* table. (The left table is the *first* table named in the FROM clause.)

The RIGHT [OUTER] JOIN will yield all rows with matching values in the join columns, plus all of the unmatched rows from the *right* table. (The right table is the *second* table named in the FROM clause.)

The FULL [OUTER] JOIN will yield all rows with matching values in the join columns, plus all the unmatched rows from both tables named in the FROM clause.

3. Using tables named T1 and T2, write a query example for each of the three join types you described in Question 2. Assume that T1 and T2 share a common column named C1.

LEFT OUTER JOIN example:

```
SELECT * FROM T1 LEFT OUTER JOIN T2 ON T1.C1 = T2.C1;
```

RIGHT OUTER JOIN example:

```
SELECT * FROM T1 RIGHT OUTER JOIN T2 ON T1.C1 = T2.C1;
```

FULL OUTER JOIN example:

```
SELECT * FROM T1 FULL OUTER JOIN T2 ON T1.C1 = T2.C1;
```

4. What is a subquery, and what are its basic characteristics?

A subquery is a query (expressed as a SELECT statement) that is located inside another query. The first SQL statement is known as the outer query, the second is known as the inner query or subquery. The inner query or subquery is normally executed first. The output of the inner query is used as the input for the outer query. A subquery is normally expressed inside parenthesis and can return zero, one, or more rows and each row can have one or more columns.

A subquery can appear in many places in a SQL statement:

- as part of a FROM clause,
- to the right of a WHERE conditional expression,
- to the right of the IN clause,
- in a EXISTS operator,
- to the right of a HAVING clause conditional operator,

- in the attribute list of a SELECT clause.

Examples of subqueries are:

```
INSERT INTO PRODUCT
SELECT * FROM P;

DELETE FROM PRODUCT
WHERE V_CODE IN (SELECT V_CODE FROM VENDOR
                 WHERE V_AREACODE = '615');

SELECT    V_CODE, V_NAME
FROM      VENDOR
WHERE     V_CODE NOT IN (SELECT V_CODE FROM PRODUCT);
```

5. **What are the three types of results a subquery can return?**
A subquery can return 1) a single value (one row, one column), 2) a list of values (many rows, one column), or 3) a virtual table (many rows, many columns).

6. **What is a correlated subquery? Give an example.**

A correlated subquery is subquery that executes once for each row in the outer query. This process is similar to the typical nested loop in a programming language. Contrast this type of subquery to the typical subquery that will execute the innermost subquery first, and then the next outer query … until the last outer query is executed. That is, the typical subquery will execute in serial order, one after another, starting with the innermost subquery. In contrast, a correlated subquery will run the outer query first, and then it will run the inner subquery once for each row returned in the outer subquery.

For example, the following subquery will list all the product line 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.)

```
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);
```

The previous nested query will execute the inner subquery once to compute the average sold units for each product code returned by the outer query.

7. **Explain the difference between a regular subquery and a correlated subquery.**
A regular, or uncorrelated subquery, executes before the outer query. It executes only once and the result is held for use by the outer query. A correlated subquery relies in part on the outer query, usually through a WHERE criteria in the subquery that

references an attribute in the outer query. Therefore, a correlated subquery will execute once for each row evaluated by the outer query; and the correlated subquery can potentially produce a different result for each row in the outer query.

8. **What does it mean to say that SQL operators are set-oriented?**
The description of SQL operators as set-oriented means that the commands work over entire tables at a time, not row-by-row.

9. **The relational set operators UNION, INTERSECT, and MINUS work properly only if the relations are union-compatible. What does *union-compatible* mean, and how would you check for this condition?**

Union compatible means that the relations yield attributes with identical names and compatible data types. That is, the relation **A(c1,c2,c3)** and the relation **B(c1,c2,c3)** have union compatibility if both relations have the same number of attributes, and corresponding attributes in the relations have "compatible" data types. *Compatible data types do not require that the attributes be exactly identical* -- only that they are *comparable*. For example, VARCHAR(15) and CHAR(15) are comparable, as are NUMBER (3,0) and INTEGER, and so on. Note that this is a practical definition of union-compatibility, which is different than the theoretical definition discussed in Chapter 3. From a theoretical perspective, corresponding attributes must have the same *domain*. However, the DBMS does not understand the meaning of the business domain so it must work with a more concrete understanding of the data in the corresponding columns. Thus, it only considers the data types.

10. **What is the difference between UNION and UNION ALL? Write the syntax for each.**

UNION yields unique rows. In other words, UNION eliminates duplicates rows. On the other hand, a UNION ALL operator will yield all rows of both relations, including duplicates. Notice that for two rows to be duplicated, they must have the same values in all columns.

To illustrate the difference between UNION and UNION ALL, let's assume two relations:

A (ID, Name) with rows (1, Lake, 2, River, and 3, Ocean)
and
B (ID, Name) with rows (1, River, 2, Lake, and 3, Ocean).

Given this description,
```
SELECT * FROM A
UNION
SELECT * FROM B
```

will yield:

ID	**Name**
1	Lake
2	River
3	Ocean
1	River
2	Lake

while

```
SELECT * FROM A
UNION ALL
SELECT * FROM B
```

will yield:

ID	Name
1	Lake
2	River
3	Ocean
1	River
2	Lake
3	Ocean

11. Suppose that you have two tables, EMPLOYEE and EMPLOYEE_1. The EMPLOYEE table contains the records for three employees: Alice Cordoza, John Cretchakov, and Anne McDonald. The EMPLOYEE_1 table contains the records for employees John Cretchakov and Mary Chen. Given that information, what is the query output for the UNION query? (List the query output.)

The query output will be:

Alice Cordoza
John Cretchakov
Anne McDonald
Mary Chen

12. Given the employee information in Question 11, what is the query output for the UNION ALL query? (List the query output.)

The query output will be:

Alice Cordoza
John Cretchakov
Anne McDonald
John Cretchakov

Mary Chen

13. Given the employee information in Question 11, what is the query output for the INTERSECT query? (List the query output.)

The query output will be:

John Cretchakov

14. Given the employee information in Question 1, what is the query output for the MINUS query? (List the query output.)

This question can yield two different answers. If you use

```
SELECT * FROM EMPLOYEE
MINUS
SELECT * FROM EMPLOYEE_1
```

the answer is

Alice Cordoza
Ann McDonald

If you use

```
SELECT * FROM EMPLOYEE_1
MINUS
SELECT * FROM EMPLOYEE
```

the answer is

Mary Chen

15. Why does the order of the operands (tables) matter in a MINUS query but not in a UNION query?

MINUS queries are analogous to algebraic subtraction – it results in the value that existed in the first operand that is not in the second operand. UNION queries are analogous to algebraic addition – it results in a combination of the two operands. (These analogies are not perfect, obviously, but they are helpful when learning the basics.) Addition and UNION have the commutative property ($a + b = b + a$), while subtraction and MINUS do not ($a – b \neq b – a$).

16. **What MS Access/SQL Server function should you use to calculate the number of days between the current date and January 25, 1999?**

    SELECT DATE()-#25-JAN-1999#

    NOTE: In MS Access you do not need to specify a FROM clause for this type of query.

17. **What Oracle function should you use to calculate the number of days between the current date and January 25, 1999?**

    ```
    SELECT    SYSDATE – TO_DATE('25-JAN-1999', 'DD-MON-YYYY')
    FROM      DUAL;
    ```

    Note that in Oracle, the SQL statement requires the use of the FROM clause. In this case, you may use the DUAL table. (The DUAL table is a dummy "virtual" table provided by Oracle for this type of query. The table contains only one row and one column so queries against it can return just one value.)

18. **Suppose that a PRODUCT table contains two attributes, PROD_CODE and VEND_CODE. Those two attributes have values of ABC, 125, DEF, 124, GHI, 124, and JKL, 123, respectively. The VENDOR table contains a single attribute, VEND_CODE, with values 123, 124, 125, and 126, respectively. (The VEND_CODE attribute in the PRODUCT table is a foreign key to the VEND_CODE in the VENDOR table.) Given that information, what would be the query output for:**

    Because the common attribute is V_CODE, the output will only show the V_CODE values generated by the each query.

    a. **A UNION query based on these two tables?**

       125,124,123,126

    b. **A UNION ALL query based on these two tables?**

       125,124,124,123,123,124,125,126

    c. **An INTERSECT query based on these two tables?**

       123,124,125

    d. **A MINUS query based on these two tables?**

       If you use PRODUCT MINUS VENDOR, the output will be NULL
       If you use VENDOR MINUS PRODUCT, the output will be 126

19. What string function should you use to list the first three characters of a company's EMP_LNAME values? Give an example, using a table named EMPLOYEE.

In Oracle, you use the SUBSTR function as illustrated next:

SELECT SUBSTR(EMP_LNAME, 1, 3) FROM EMPLOYEE;

In SQL Server, you use the SUBSTRING function as shown:

SELECT SUBSTRING(EMP_LNAME, 1, 3) FROM EMPLOYEE;

20. What is a sequence? Write its syntax.

A sequence is a special type of object that generates unique numeric values in ascending or descending order. You can use a sequence to assign values to a primary key field in a table.

A sequence provides functionality *similar* to the Autonumber data type in MS Access. For example, both, sequences and Autonumber data types provide unique ascending or descending values. However, there are some subtle differences between the two:

- In MS Access an Autonumber is a *data type*; in Oracle a sequence is *a completely independent object*, rather than a data type.
- In MS Access, you can only have one Autonumber per table; in Oracle you can have as many sequences as you want and they are not tied to any particular table.
- In MS Access, the Autonumber data type is tied to a field in a table; in Oracle, the sequence-generated value is not tied to any field in any table and can, therefore, be used on any attribute in any table.

The syntax used to create a sequence is:

CREATE SEQUENCE CUS_NUM_SEQ START WITH 100 INCREMENT BY 10 NOCACHE;

21. What is a trigger, and what is its purpose? Give an example.

A trigger is a block of PL/SQL code that is automatically invoked by the DBMS upon the occurrence of a data manipulation event (INSERT, UPDATE or DELETE.) Triggers are always associated with a table and are invoked before or after a data row is inserted, updated, or deleted. Any table can have one or more triggers.

Triggers provide a method of enforcing business rules such as:

- A customer making a credit purchase must have an active account.
- A student taking a class with a prerequisite must have completed that prerequisite with a B grade.

- To be scheduled for a flight, a pilot must have a valid medical certificate and a valid training completion record.

Triggers are also excellent for enforcing data constraints that cannot be directly enforced by the data model. For example, suppose that you must enforce the following business rule:

> If the quantity on hand of a product falls below the minimum quantity, the P_REORDER attribute must the automatically set to 1.

To enforce this business rule, you can create the following TRG_PRODUCT_REORDER trigger:

```
CREATE OR REPLACE TRIGGER TRG_PRODUCT_REORDER
BEFORE INSERT OR UPDATE OF P_ONHAND, P_MIN ON PRODUCT
FOR EACH ROW
BEGIN
    IF :NEW.P_ONHAND <= :NEW.P_MIN THEN
    NEW.P_REORDER := 1;
    ELSE
        :NEW.P_REORDER := 0;
    END IF;
END;
```

22. What is a stored procedure, and why is it particularly useful? Give an example.

A stored procedure is a named block of PL/SQL and SQL statements. One of the major advantages of stored procedures is that they can be used to encapsulate and represent business transactions. For example, you can create a stored procedure to represent a product sale, a credit update, or the addition of a new customer. You can encapsulate SQL statements within a single stored procedure and execute them as a single transaction.

There are two clear advantages to the use of stored procedures:

1. Stored procedures substantially reduce network traffic and increase performance. Because the stored procedure is stored at the server, there is no transmission of individual SQL statements over the network.
2. Stored procedures help reduce code duplication through code isolation and code sharing (creating unique PL/SQL modules that are called by application programs), thereby minimizing the chance of errors and the cost of application development and maintenance.

For example, the following PRC_LINE_ADD stored procedure will add a new invoice line to the LINE table and it will automatically retrieve the correct price from the PRODUCT table.

```
CREATE OR REPLACE PROCEDURE PRC_LINE_ADD
```

```
            (W_LN IN NUMBER, W_P_CODE IN VARCHAR2, W_LU NUMBER)
AS
    W_LP NUMBER := 0.00;
BEGIN
    -- GET THE PRODUCT PRICE
    SELECT P_PRICE INTO W_LP
        FROM PRODUCT
            WHERE P_CODE = W_P_CODE;

    -- ADDS THE NEW LINE ROW
    INSERT INTO LINE
    VALUES(INV_NUMBER_SEQ.CURRVAL, W_LN, W_P_CODE, W_LU,
W_LP);

    DBMS_OUTPUT.PUT_LINE('Invoice line ' || W_LN || ' added');
END;
```

23. What is embedded SQL, and how is it used?

Embedded SQL is a term used to refer to SQL statements that are contained within an application programming language such as COBOL, C++, ASP, Java, or ColdFusion. The program may be a standard binary executable in Windows or Linux, or it may be a Web application designed to run over the Internet. No matter what language you use, if it contains embedded SQL statements it is called the *host language*. Embedded SQL is still the most common approach to maintaining procedural capabilities in DBMS-based applications.

For example, the following embedded SQL code will delete the employee 109, George Smith, from the EMPLOYEE table:

```
EXEC SQL
DELETE FROM EMPLOYEE WHERE EMP_NUM = 109;
END-EXEC.
```

Remember that the preceding embedded SQL statement is compiled to generate an executable statement. Therefore, the statement is fixed permanently; that is, it cannot change -- unless, of course, the programmer changes it. Each time the program runs, it deletes the same row. *In short, the preceding code is good only for the first run; all subsequent runs will more than likely give an error*. (Employee 109 is deleted the first time the embedded statement is executed. If you run the code again, it tries to delete an employee who has already been deleted.) Clearly, this code would be more useful if you are able to specify a variable to indicate which employee number is to be deleted.

24. What is dynamic SQL, and how does it differ from static SQL?

Dynamic SQL is a term used to describe an environment in which the SQL statement is not known in advance; instead, the SQL statement is generated at run time. In a dynamic SQL environment, a program can generate the SQL statements (at run time) that are required to respond to ad-hoc queries. In such an environment, neither the programmers nor end users are likely to know precisely what kind of queries are to be generated or how those queries are to be structured. For example, a dynamic SQL equivalent of the example shown in question 19 might be:

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

Note that the attribute list and the condition are not known until the end user specifies them. W_TABLE, W_ATRIBUTE_LIST and W_CONDITION are text variables that contain the end-user input values used in the query generation. Because the program uses the end user input to build the text variables, the end user can run the same program multiple times to generate different outputs. For example, in one case the end user may one to know what products have a price less than $100; in another case, the end user may want to know how many units of a given product are available for sale at any given moment.

Although dynamic SQL is clearly flexible, such flexibility carries a price. Dynamic SQL tends to be much slower that static SQL and it requires more computer resources (overhead). In addition, you are more likely to find different levels of support and incompatibilities between DBMS vendors.