**Chapter 7**

**Introduction to Structured Query Language (SQL)**

|  |
| --- |
| **NOTE**  Several points are worth emphasizing:   * We have provided the SQL scripts for both chapters 7 and 8. 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. * Because this chapter focuses on learning SQL, we recommend that if you use Microsoft Access, that you use the Microsoft Access SQL window to type SQL queries. Using this approach, you will be able to demonstrate the interoperability of standard SQL. For example, you can cut and paste the same SQL command from the SQL query window in Microsoft Access, to Oracle SQL \* Plus and to MS SQL Query Analyzer. This approach achieves two objectives:   + It demonstrates that adhering to the SQL standard means that most of the SQL code will be portable among DBMSes.   + It also demonstrates that even a widely accepted SQL standard is sometimes implemented with slight distinctions by different vendors. For example, the treatment of date formats in Microsoft Access and Oracle is slightly different. * Chapter 7 is all about SELECT queries to retrieve data. We choose to start with SELECT queries because simple SELECT queries are conceptually easy to understand, which gives students a good place to start. Also, most database jobs will require students to work with databases that are already in place. We emphasize to students the importance to learning the data model in which they work. This also provides an opportunity to highlight the importance of good naming conventions when creating the database design. Students can see how helpful it is to have a proper naming convention for attributes within an entity, the importance of having the name of the foreign key reflect the table from which it originates, and the benefits of descriptive entity and attribute names. |

**Answers to Review Questions**

**1. Explain why it would be preferable to use a DATE data type to store date data instead of a character data type.**

The DATE data type uses numeric values based on the Julian calendar to store dates. This makes date arithmetic such as adding and subtracting days or fractions of days possible (as well as numerous special date-oriented functions discussed in the next chapter!).

**2. Explain why the following command would create an error, and what changes could be made to fix the error.**

**SELECT V\_CODE, SUM(P\_QOH) FROM PRODUCT;**

The command would generate an error because an aggregate function is applied to the P\_QOH attribute but V\_CODE is neither in an aggregate function nor in a GROUP BY clause. This can be fixed by either 1) placing V\_CODE in an appropriate aggregate function based on the data that is being requested by the user, 2) adding a GROUP BY clause to group by values of V\_CODE (i.e. GROUP BY V\_CODE), 3) removing the V\_CODE attribute from the SELECT clause, or 4) removing the Sum aggregate function from P\_QOH. Which of these solutions is most appropriate depends on the question that the query was intended to answer.

**3. 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.

**4. 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.

**5. 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;

**6. What is a recursive join?**

A recursive join is a join in which a table is joined to itself.

**7. Rewrite the following WHERE clause without the use of the IN special operator.**

**WHERE V\_STATE IN (‘TN’, ‘FL’, ‘GA’)**

WHERE V\_STATE = 'TN' OR V\_STATE = 'FL' OR V\_STATE = 'GA'

Notice that each criteria must be complete (i.e. attribute-operator-value).

**8. Explain the difference between an ORDER BY clause and a GROUP BY clause.**

An ORDER BY clause has no impact on which rows are returned by the query, it simply sorts those rows into the specified order. A GROUP BY clause does impact the rows that are returned by the query. A GROUP BY clause gathers rows into collections that can be acted on by aggregate functions.

**9. Explain why the two following commands produce different results.**

**SELECT DISTINCT COUNT (V\_CODE) FROM PRODUCT;**

**SELECT COUNT (DISTINCT V\_CODE) FROM PRODUCT;**

The difference is in the order of operations. The first command executes the Count function to count the number of values in V\_CODE (say the count returns "14" for example) including duplicate values, and then the Distinct keyword only allows one count of that value to be displayed (only one row with the value "14" appears as the result). The second command applies the Distinct keyword to the V\_CODEs before the count is taken so only unique values are counted.

**10. What is the difference between the COUNT aggregate function and the SUM aggregate function?**

COUNT returns the number of values without regard to what the values are. SUM adds the values together and can only be applied to numeric values.

**11. In a SELECT query, what is the difference between a WHERE clause and a HAVING clause?**

Both a WHERE clause and a HAVING clause can be used to eliminate rows from the results of a query. The differences are 1) the WHERE clause eliminates rows before any grouping for aggregate functions occurs while the HAVING clause eliminates groups after the grouping has been done, and 2) the WHERE clause cannot contain an aggregate function but the HAVING clause can.

**12. 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);

**13. 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).

**14. 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.

**15. 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.

**16. 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.

**17. 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.

**18. 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
4. River
5. Lake

while

SELECT \* FROM A

UNION ALL

SELECT \* FROM B

will yield:

ID Name

1. Lake
2. River
3. Ocean
4. River
5. Lake
6. Ocean

**19. 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

**20. Given the employee information in Question 19, 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

**21. Given the employee information in Question 19, what is the query output for the INTERSECT query? (List the query output.)**

The query output will be:

John Cretchakov

**22. Given the employee information in Question 19, 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

**23. 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.

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

125,124,123,126

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

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

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

123,124,125

1. **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

**24. 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 ≠ b – a).

**25. 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.

**26. What Oracle function should you use to calculate the number of days between your birthday and the current date?**

The SYSDATE keyword can be used to retrieve the current date from the server. By subtracting your birthdate from the current date, using date arithmetic, the number of dates will be returned. 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.)

**27. 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;

**28. What two things must a SQL programmer understand before beginning to craft a SELECT query?**

Before crafting a SELECT query, the SQL programmer must 1) understand the data model in which the query will operate, and 2) the problem being solved. Data models are often complex to the point that knowing what data is available, the meaning of that data, and how to transform the data to produce the desired results will require the programmer to become very familiar with the data model before the query can be created. Problem statements that seem clear to users can often be interpreted in many ways, so it is important for the programmer to understand exactly what the user is requesting.

**Problem Solutions**

All of the problems in the Problem section require writing SQL code. Since there are minor differences in the code based on the DBMS used, solutions for all of the problems are provided in separate files for Oracle, MySQL, and Microsoft SQL Server. Solutions for Microsoft Access are provided in .mdb files for each data model used in the problem section. The files are located in the “**Teacher**” data files that accompany the book, and are named as follows:

**Oracle: Ch07\_ProblemSolutions\_ORA.txt**

**MySQL: Ch07\_ProblemSolutions\_MySQL.txt**

**SQL Server: Ch07\_ProblemSolutions\_SQL.txt**

**MS Access: Ch07\_ConstructCo.mdb**

**Ch07\_Fact.mdb**

**Ch07\_LargeCo.mdb**

**Ch07\_SaleCo.mdb**