SELF TEST ANSWERS

Define Subqueries

1. ☑ **A, B, C, D, E.** Subqueries can be used at all these points.
   ☒ **F.** A subquery cannot be used in the ORDER BY clause of a query.

2. ☑ **D.** Subquery nesting can be done to many levels.
   ☒ **A, B,** and **C. A** and **C** are wrong because subqueries can be nested. **B** is wrong because the number of rows returned is not relevant to nesting subqueries, only to the operators being used.

3. ☑ **A.** The result set of the inner query is needed before the outer query can run.
   ☒ **B, C,** and **D. B** and **C** are not possible because the result of the subquery is needed before the parent query can start. **D** is wrong because the subquery is only run once.

4. ☑ **D.** This is a correlated subquery, which must be run for every row in the table.
   ☒ **A, B,** and **C.** The result of the inner query is dependent on a value from the outer query; it must therefore be run once for every row.

Describe the Types of Problems That the Subqueries Can Solve

5. ☑ **A, D.** The two statements will deliver the same result, and neither will fail if the name is duplicated.
   ☒ **B, C. B** is wrong because the statements are functionally identical, though syntactically different. **C** is wrong because the comparison operator used, IN, can handle a multiple-row subquery.

List the Types of Subqueries

6. ☑ **A, B.** A scalar subquery can be defined as a query that returns a single value.
   ☒ **C, D. C** is wrong because a scalar subquery is the only subquery that can be used in the SELECT LIST. **D** is wrong because scalar subqueries can be correlated.

7. ☑ **E.** ALL, ANY, IN, and NOT IN are the multiple-row comparison operators.
   ☒ **A, B, C, D.** All of these can be used.

Write Single-Row and Multiple-Row Subqueries

8. ☑ **C.** The equality operator requires a single-row subquery, and the second subquery could return several rows.
   ☒ **A, B, D. A** is wrong because the statement will fail in all circumstances except the unlikely case where there is zero or one employees. **B** is wrong because this is not a problem; there need be no relationship between the source of data for the inner and outer queries. **D** is wrong because the subquery will only run once; it is not a correlated subquery.

9. ☑ **A** and **B** are identical.
☒ **C** is logically the same as **A** and **B** but syntactically is not possible; it will give an error.
D will always return no rows, because it asks for all employees who have a salary lower than all employees. This is not an error but can never return any rows. The filter on DEPARTMENTS is not relevant.

10. ☑ **C.** If a subquery returns NULL, then the comparison will also return NULL, meaning that no rows will be retrieved.
☒ **A, B, D. A** is wrong because this would not cause an error. **B** is wrong because a comparison with NULL will return nothing, not everything. **D** is wrong because a comparison with NULL can never return anything, not even other NULLs.

LAB ANSWER

The following are two possible solutions using ANY and MIN:

```
select last_name from employees where
salary > any (select salary from employees where last_name='Taylor')
order by last_name;

select last_name from employees where
salary not < (select min(salary) from employees where last_name='Taylor')
order by last_name;
```

These are just as valid as the solutions presented earlier that used ALL and MAX, but they do not give the same result. There is no way to say that these are better or worse than the earlier solutions. The problem is that the subquery is based on a column that is not the primary key. It would not be unreasonable to say that all these solutions are wrong, and the original query is the best; it gives a result that is unambiguously correct if the LAST_NAME is unique, and if LAST_NAME is not unique, it throws an error rather than giving a questionable answer. The real answer is that the query should be based on EMPLOYEE_ID, not LAST_NAME.