Item: 1 (Ref:Cert-1Z0-071.7.4.2)

Consider the following query executed for an `employees` table:

```
SELECT * FROM employees WHERE (dept_id, salary) IN (SELECT deptid, MAX(Salary) FROM employees
WHERE jobprofile != 'Manager' GROUP BY deptid);
```

What is the output of the given query?

- ◯ Details of all the employees are displayed.
- ◯ Details of all the employees are displayed department-wise.
- ◯ Details of the employee having the highest salary across all departments are displayed.
- ◯ Details of the employees having the highest salary in each department are displayed.

Answer:

Details of the employees having the highest salary in each department are displayed.

Explanation:

The given query displays the details of the employees having the highest salary in each department. The subquery returns the maximum salary in each department for only those employees who are not managers. Then `IN` operator then checks if a row with the department number and maximum salary is in the result set of the subquery. The query then displays all those rows with the maximum salary.

It is incorrect that the given query displays the details of all the employees. The details of all the employees will be displayed by the following query:

```
SELECT * FROM employees;
```

It is incorrect that the query displays details of all the employees department-wise. Details of the employees for each department will be displayed by the following query:

```
SELECT * FROM employees GROUP BY deptid;
```

It is incorrect that the query displays details of the employee having the highest salary across all departments. That information could be determined from the following query:

```
SELECT * FROM employees WHERE salary = (SELECT MAX(salary) FROM employees);
```

Item: 2 (Ref:Cert-1Z0-071.7.8.2)

Evaluate this `SELECT` statement:

```
SELECT s.student_name, s.grade_point_avg, s.major_id, m.gpa_avg
FROM student s, (SELECT major_id, AVG(grade_point_avg) gpa_avg
FROM student
GROUP BY major_id) m
WHERE s.major_id = m.major_id AND s.grade_point_avg > m.gpa_avg;
```

What will be the result of executing this `SELECT` statement?

- ◯ The names of all students with a grade point average that is higher than the average grade point average in their major will be displayed.
- ◯ The names of all students with a grade point average that is higher than the average grade point average of all students will be displayed.
- ◯ The names of all students, grouped by each major, with a grade point average that is higher than the average grade point average of all students in each major will be displayed.
- ◯ A syntax error will occur because of ambiguous table aliases.

◯ A syntax error will be returned because the FROM clause cannot contain a subquery.

Answer:

The names of all students with a grade point average that is higher than the average grade point average in their major will be displayed.

Explanation:

The names of all students with a grade point average that is higher than the average grade point average in their major will be displayed. You can use a subquery in the FROM clause of a SELECT statement to define a data source for the SELECT statement. This is helpful if you need to view aggregate values but need to include columns in the select list that are not grouped. In this scenario, the subquery returns the major_id values and the average grade point average of students with each major. This result is then used as a data source for the main query. The WHERE clause of the main query joins this result table to the STUDENT table using major_id and ensures that you only return rows where the grade point average is higher than the average grade point average of the student's major.

The option stating that the names of all students with a grade point average that is higher than the average grade point average of all students will be displayed is incorrect. Because the inner query groups the records by major and the outer query WHERE clause joins on major, each student's grade point average will be compared to the average grade point average for the same major.

The option stating that the names of all students grouped by major are displayed is incorrect. The outer query does not contain a GROUP BY clause, so records returned are not grouped.

The option stating that an error will occur because of ambiguous table aliases is incorrect. Although the table alias m is used twice, this is acceptable. The main query uses the table alias for the entire subquery.

The option stating that an error will occur because the FROM clause cannot contain a subquery is incorrect because a table, view, or subquery is allowed in the FROM clause of a SELECT statement.

Item: 3 (Ref:Cert-1Z0-071.7.1.1)

Which two statements regarding the valid use of single-row and multiple-row subqueries are true? (Choose two.)

- ☐ Single-row subqueries can only be used in a WHERE clause.
- ☐ Multiple-row subqueries can be used with the LIKE operator.
- ☐ Single-row operators can only be used with single-row subqueries.
- ☐ Single- and multiple-row subqueries can be used with the BETWEEN operator.
- ☐ Multiple-row subqueries can be used with both single-row and multiple-row operators.
- ☐ Multiple-row subqueries can be used in a WHERE clause and the INTO portion of an INSERT statement.

Answer:

Single-row operators can only be used with single-row subqueries.

Multiple-row subqueries can be used in a WHERE clause and the INTO portion of an INSERT statement.

Explanation:

The following two statements regarding the valid use of single-row and multiple-row subqueries are true:

- Single-row operators can only be used with single-row subqueries.
- Multiple-row subqueries can be used in a WHERE clause and the INTO portion of an INSERT statement.

A single-row subquery is a subquery that returns only one row from the inner SELECT statement. Single-row subqueries can only be used with single-row operators, such as =, > , >=, <, <=, or <>. When a single-row operator is used, then the subquery must be a single-row subquery that returns only one row. If you attempt to use a single-row operator with a subquery that returns multiple rows,

an error occurs. Multiple-row subqueries can be used in a WHERE clause and the INTO portion of an INSERT statement. When used in a WHERE clause, the multiple-row subquery must use a multiple-row operator, such as IN, ANY, or ALL. When used in the INTO portion of an INSERT statement, all rows returned by the multiple-row subquery are inserted into the specified table.

The option stating that single-row subqueries can only be used in a WHERE clause is incorrect. Single-row subqueries can be used any place that a single scalar value can be used.

The option stating that multiple-row subqueries can be used with the LIKE operator is incorrect. The LIKE operator accepts a single value and can only be used with single-row queries.

The option stating that single- and multiple-row subqueries can be used with the BETWEEN operator is incorrect. The BETWEEN operator accepts two values. Either one or both of these values may come from single-row subqueries. However, the values may not come from multiple-row subqueries.

The option stating that multiple-row subqueries can be used with both single-row and multiple-row operators is incorrect because they can only use multiple-row operators.

Item: 4 (Ref:Cert-1Z0-071.7.8.7)

Evaluate this SQL statement:

```
SELECT product_id, product_name, price
FROM product
WHERE supplier_id IN (SELECT supplier_id
FROM product
WHERE price > 120 OR qty_in_stock > 100);
```

Which values will be displayed?

- ◯ the product_id, product_name, and price of products that are priced greater than $120.00 and have a qty_in_stock value greater than 100
- ◯ the product_id, product_name, and price of products that are priced greater than $120.00 or that have a qty_in_stock value greater than 100
- ◯ the product_id, product_name, and price of products that are priced greater than $120.00 or that have a qty_in_stock value greater than 100, and that have a supplier
- ◯ the product_id, product_name, and price of products supplied by a supplier with products that are priced greater than $120.00 or with products that have a qty_in_stock value greater than 100

Answer:

the product_id, product_name, and price of products supplied by a supplier with products that are priced greater than $120.00 or with products that have a qty_in_stock value greater than 100

Explanation:

The subquery will return the supplier_id values of products that have a price value greater than $120.00 or have a qty_in_stock value greater than 100. The main query will return the product_id, product_name, and price values for all products with supplier_id values equal to those returned by the subquery.

The query does not return the product_id, product_name, and price of products that are priced greater than $120.00 and have a qty_in_stock value greater than 100. To accomplish this, you would use the following query:

```
SELECT product_id, product_name, price
FROM product
WHERE price > 120 AND qty_in_stock > 100;
```

The query does not return the product_id, product_name, and price of products that are priced greater than $120.00 or that have a qty_in_stock value greater than 100. To accomplish this, you would use the following query:

```
SELECT product_id, product_name, price
FROM product
```

```
WHERE price > 120 OR qty_in_stock > 100;
```

The query does not return the product_id, product_name, and price of products that are priced greater than $120.00 or that have a qty_in_stock value greater than 100, and that have a supplier. To accomplish this, you would use the following query:

```
SELECT product_id, product_name, price
FROM product
WHERE (price > 120 OR qty_in_stock > 100)
AND supplier_id IS NOT NULL;
```

Item: 5 (Ref:Cert-1Z0-071.7.6.1)

You are the database administrator for a corporate organization. You need to perform the following database operations:

- Update the designation column in the employees table for those managers for whom the productivity_score is greater than 85 in the productivity table.
- Insert a new row in the productivity_score table from the employees table for those employees who have completed working for six months. Any employees whose ID is not yet assigned also need to be considered for insertion.

Which of the following statements about the operators is TRUE to achieve the desired results in the most efficient way? (Choose all that apply).

- ☐ The EXISTS operator in the UPDATE statement.
- ☐ The NOT EXISTS operator in the INSERT statement.
- ☐ The IN operator in the UPDATE statement.
- ☐ The NOT IN operator in the INSERT statement.

Answer:

The EXISTS operator in the UPDATE statement.

The NOT EXISTS operator in the INSERT statement.

Explanation:

In this scenario, the following two statements are true to achieve the desired results in the most efficient way:

- The EXISTS operator in the UPDATE statement.
- The NOT EXISTS operator in the INSERT statement.

For updating the rows in the Employees table, you should use the EXISTS operator as it checks whether the rows returned by the subquery exists in the rows returned by the outer query. If there is at least one row returned by the subquery, the EXISTS operator returns TRUE; otherwise, the operator returns FALSE. The EXISTS operator can be used in this scenario as shown below:

```
UPDATE employees e
SET designation='Senior Manager'
WHERE EXISTS (SELECT emp_id FROM productivity p WHERE e.emp_id=p.emp_id AND
p.productivity_score>85)
```

The subquery in the UPDATE statement returns those rows from the productivity table for which employee ID matches the employee ID of the managers and score of those managers is more than 85. If the subquery returns at least one row, then that row is updated in the employees table.

For inserting rows in the productivity table from the employees table, you should use the NOT EXISTS operator. The NOT EXISTS operator is logically opposite of the EXISTS operator, that is, it returns TRUE for those rows in the productivity table such that there no rows returned by the subquery for which the employees have completed six months. The NOT EXISTS operator can be used in this scenario as shown below:

```
INSERT INTO productivity
```

```
(SELECT emp_id, score FROM productivity p
WHERE NOT EXISTS (SELECT emp_id FROM employees e
WHERE e.emp_id=p.emp_id AND duration>6));
```

The options stating that you should use the `IN` operator in the `UPDATE` statement and the `NOT IN` operator in the `INSERT` statement are incorrect. This is because the `IN` operator is less efficient than the `EXISTS` operator, and the `NOT IN` operator returns `FALSE` for rows that have `NULL` values. This means that employees who do not have an ID are ignored by the `NOT IN` operator.

Item: 6 (Ref:Cert-1Z0-071.7.1.3)

Which statement regarding subqueries is true?

- ◯ Subqueries can return multiple columns.
- ◯ Subqueries can be nested up to five levels.
- ◯ A subquery must be placed on the right side of the comparison operator.
- ◯ A subquery cannot reference a table that is not included in the outer query's `FROM` clause.

Answer:

Subqueries can return multiple columns.

Explanation:

Subqueries can return multiple columns. Subqueries that return multiple columns are referred to as multiple-column subqueries. Multiple-column subqueries are more commonly used as data sources in the `FROM` clause of an outer `SELECT` statement but can also be used to compare multiple columns in a `WHERE` clause. The syntax of a `SELECT` statement performing such a comparison is:

```
SELECT column, column, column,
FROM table1
WHERE (column, column, ...) IN
(SELECT column, column,
FROM table2
WHERE condition);
```

The option stating that subqueries can be nested up to five levels is incorrect because no such limit exists. The only limitation to creating queries is that of available memory. Subqueries are often placed on the right side of a comparison operator for readability, but this is not required.

A subquery can access different tables than the outer query that contains it. Therefore, the option stating that a subquery cannot reference a table that is not included in the outer query's `FROM` clause is incorrect.

Item: 7 (Ref:Cert-1Z0-071.7.5.2)

You are the database administrator in a corporate organization. Records of all employees are stored in the `employees` table, while records of all retired employees are stored in the `retired_employees` table. You need to remove the rows from the `employees` table for all the employees who have retired before 1990.

Which of the following statements should you use to achieve the desired results?

- ◯
```
DELETE FROM retired_employees

WHERE retire_year<1990;
```
- ◯
```
DELETE FROM retired_employees
WHERE emp_id IN (SELECT emp_id FROM employees);
```
- ◯
```
DELETE FROM employees
WHERE emp_id IN (SELECT emp_id FROM retired_employees

WHERE retire_year<1990);
```

○
```
DELETE FROM employees
WHERE emp_id NOT IN (SELECT emp_id FROM retired_employees

WHERE retire_year<1990);
```

Answer:

```
DELETE FROM employees
WHERE emp_id IN (SELECT emp_id FROM retired_employees

WHERE retire_year<1990);
```

Explanation:

You should use the following statement to delete the rows of those retired employees that have retired before 1990:

```
DELETE FROM employees
WHERE emp_id IN (SELECT emp_id FROM retired_employees
WHERE retire_year<1990);
```

The `DELETE` statement has a subquery that returns the `emp_id` of those rows from the `retired_employees` for which the `retire_year` column is less than 1990. The `IN` operator then checks whether there are any employees in the `employees` table that are also returned by the subquery. If there are any such employees, then their records are deleted from the `employees` table.

You should not use the following statement in this scenario:

```
DELETE FROM retired_employees
WHERE retire_year<1990;
```

This statement deletes rows from the `retired_employees` for those employees who have retired before 1990. The rows are deleted from the `retired_employees` instead of the `employees` table.

The following statement does not achieve the desired results:

```
DELETE FROM retired_employees
WHERE emp_id IN (SELECT emp_id FROM employees);
```

This statement deletes rows from the `retired_employees` table for which `emp_id` exists in the `employees` table. This means records of those retired employees who have an ID in the `employees` table are deleted. This will result in deleting the records of all the retired employees, including those employees who retired after 1990.

The following statement does not display the desired output:

```
DELETE FROM employees
WHERE emp_id NOT IN (SELECT emp_id FROM retired_employees
WHERE retire_year<1990);
```

This statement deletes rows from the `employees` table for all the employees except those who have retired before 1990. This because the `NOT IN` operator excludes all the rows of the `employees` table who have retired before 1990. Therefore, the `WHERE` condition is true only for those rows where employees either are still working or retired after 1990.

Item: 8 (Ref:Cert-1Z0-071.7.2.1)

Which construct can be used to return data based on an unknown condition?

○ a subquery

○ a `GROUP BY` clause

○ an `ORDER BY` clause

○ a `WHERE` clause with an `OR` condition

Answer:

a subquery

Explanation:
A subquery can be used to return data based on an unknown condition. Often when the condition for a query cannot be stated directly, the query can be broken into two smaller queries to return the desired result. The subquery, or inner query, returns a value that is used by the main, or outer, query.

A `GROUP BY` clause creates groups of data so that aggregate calculations, such as sums and averages, can be performed on the group. An `ORDER BY` clause sorts the results of a query based on a specified sort order. A `WHERE` clause, whether it includes a logical conditional operator or not, defines a condition that must be met for rows to be returned. None of these constructs will allow you to return data based on an unknown condition.

Item: 9 (Ref:Cert-1Z0-071.7.1.2)

How many values could a subquery used with the <> operator return?

- ○ only one
- ○ up to two
- ○ up to ten
- ○ unlimited

Answer:

only one

Explanation:
The not equal (<>) operator is a single-row operator. Single-row operators can only be used with single-row subqueries, or inner queries, that return only one row. Attempting to use the not equal (<>) operator with a query that returns more than one row will generate an error.

All other options that indicate more than one value could be returned are incorrect. If a subquery returns more than one row, then it can only be used with a multiple-row operator. Multiple-row operators include the `IN`, `ANY`, and `ALL` operators.

Item: 10 (Ref:Cert-1Z0-071.7.7.1)

Which of the following statements is FALSE about the `WITH` clause? (Choose all that apply.)

- ☐ A single `WITH` clause can contain only one query.
- ☐ The result set of the queries in a `WITH` clause is stored in the user's temporary tablespace.
- ☐ The `WITH` clause improves the performance of the queries.
- ☐ A `WITH` clause can occur inside another `WITH` clause. .

Answer:

A single `WITH` clause can contain only one query.

A `WITH` clause can occur inside another `WITH` clause. .

Explanation:
It is false to state that a single WITH clause can contain only one query and that a WITH clause can occur inside another WITH clause. The WITH clause allows the reuse of those queries that are used more than once in a complex query. This clause can contain more than one query, with each query having a distinct name with which it is referenced in the complex query.

The WITH clause cannot contain another WITH clause because nesting of WITH clauses is not supported. If you use a WITH clause inside another WITH clause, an error occurs.
It is true that the result set of the queries in a WITH clause is stored in the user's temporary tablespace. Since the queries in a WITH clause can be executed multiple times in a complex query, they are executed once and their result sets are stored temporarily in the user's tablespace.

It is true that the WITH clause improves the performance of the queries. The WITH clause allows you to reuse a set of queries that are executed multiple times in a SELECT statement. The queries in the WITH clause are executed and then stored in the user's temporary tablespace. In this way, the WITH clause improves the performance by reducing the time taken to execute the queries.

Item: 11 (Ref:Cert-1Z0-071.7.7.2)

Which of the following Oracle statements can contain a WITH clause?

- ○ INSERT
- ○ SELECT
- ○ UPDATE
- ○ DELETE

Answer:
SELECT

Explanation:
The WITH clause can be used with the SELECT statement only. This clause allows you to specify and name the queries that are frequently used in a particular SELECT statement. The queries in the WITH clause are then referenced in the SELECT statement by name, providing an easy and simplified way of using frequently used queries.

You cannot use the WITH clause with the INSERT, UPDATE, and DELETE statements because doing so is syntactically incorrect.

Item: 12 (Ref:Cert-1Z0-071.7.8.1)

Examine the data from the DONATION table.

DONATION (PLEDGE_ID is the primary key)

PLEDGE_ID	DONOR_ID	PLEDGE_DT	AMOUNT_PLEDGED	AMOUNT_PAID	PAYMENT_DATE
1	1	10-SEP-2011	1000	1000	02-OCT-2001
2	1	22-FEB-2011	1000		
3	2	08-OCT-2001	10	10	28-OCT-2001
4	2	10-DEC-2001	50		
5	3	02-NOV-2001	10000	9000	28-DEC-2001
6	3	05-JAN-2002	1000	1000	31-JAN-2002
7	4	09-NOV-2001	2100	2100	15-DEC-2001
8	5	09-DEC-2001	110	110	29-DEC-2001

This statement fails when executed:

```
SELECT amount_pledged, amount_paid
FROM donation
WHERE donor_id =
(SELECT donor_id
FROM donation
WHERE amount_pledged = 1000.00
OR pledge_dt = '05-JAN-2002');
```

Which two changes could correct the problem? (Choose two. Each correct answer is a separate solution.)

- ☐ Remove the subquery WHERE clause.
- ☐ Change the outer query WHERE clause to `WHERE donor_id IN`.
- ☐ Change the outer query WHERE clause to `WHERE donor_id LIKE`.
- ☐ Include the `donor_id` column in the select list of the outer query.
- ☐ Remove the single quotes around the date value in the inner query WHERE clause.
- ☐ Change the subquery WHERE clause to `WHERE amount_pledged = 1000.00 AND pledge_dt = '05-JAN-2002'`.

Answer:

Change the outer query `WHERE` clause to `WHERE donor_id IN`.

Change the subquery `WHERE` clause to `WHERE amount_pledged = 1000.00 AND pledge_dt = '05-JAN-2002'`.

Explanation:

This statement fails because the subquery returns multiple rows, which cannot be compared to a single value using the equality operator (=) in the outer query. To correct the problem, you could change the outer query WHERE clause to `WHERE donor_id IN`. Changing the outer query WHERE clause to use the `IN` operator would allow the inner query to return multiple rows without generating an error.

Alternatively, you could change the subquery WHERE clause to `WHERE amount_pledged = 1000.00 AND pledge_dt = '05-JAN-2002'`. Based on the given data, this change would cause only one row to be returned and would also eliminate the error.

Removing the subquery WHERE clause, changing the outer query WHERE clause to `WHERE donor_id LIKE`, or including the `donor_id` column in the select list of the outer query would not correct the problem. The inner query would still return multiple rows and produce an error.

Removing the single quotes around the date value in the inner query WHERE clause will not correct the problem. When dates values are used in a WHERE clause, they must be enclosed in single quotation marks.

Item: 13 (Ref:Cert-1Z0-071.7.4.1)

Which of the following statements are TRUE about correlated subqueries? (Choose all that apply.)

- ☐ The subquery can reference the columns specified in the parent query.
- ☐ The parent query can reference the columns specified in the subquery.
- ☐ The subquery is executed for each row returned by the parent query.
- ☐ The parent query is executed for each of the rows returned by the subquery

Answer:

The subquery can reference the columns specified in the parent query.

The subquery is executed for each row returned by the parent query.

Explanation:

The correct answer is that the subquery can reference the columns specified in the parent query and that the subquery is executed for each row returned by the parent query.

Correlated subqueries can reference the columns in the parent query to compute their result sets. These subqueries are primarily used to return a result set that depends on the column values in the result set of the parent query. The subquery computes its result set by being executed for all the rows in the result set of the parent query. After the subquery is executed for all the rows, the final result set is then returned.

It is false to state that the parent query can reference the columns specified in the subquery. The parent query cannot reference the columns in the correlated subquery; however, the subquery can reference the columns in the parent query.

It is false to state that the parent query is executed for each of the rows returned by the subquery. The parent query is executed only once to return its result set. For each row in the result set of the parent query, the correlated subquery is executed.

Item: 14 (Ref:Cert-1Z0-071.7.2.3)

Examine the structures of the CUSTOMER and CURR_ORDER tables:

```
CUSTOMER
--------------------
CUSTOMER_ID  NUMBER(5)
NAME         VARCHAR2(25)
CREDIT_LIMIT   NUMBER(8,2)
ACCT_OPEN_DATE DATE

CURR_ORDER
------------------------
ORDER_ID    NUMBER(5)
CUSTOMER_ID NUMBER(5)
ORDER_DATE  DATE
TOTAL       NUMBER(8,2)
```

Which scenario would require a subquery to return the desired results?

- ◯ You need to display the names of all the customers who placed an order today.
- ◯ You need to determine the number of orders placed this year by the customer with CUSTOMER_ID value 30450.
- ◯ You need to determine the average credit limit of all the customers who opened an account this year.
- ◯ You need to determine which customers have placed orders with amount totals larger than the average order amount.

Answer:

You need to determine which customers have placed orders with amount totals larger than the average order amount.

Explanation:

With the given tables, a subquery would be required to determine which customers have placed orders with amount totals larger than the average order amount. To return the desired result, an inner query would return the average order amount. Then, the outer query would use this value in the WHERE clause to restrict the rows returned to only those customers who had placed orders with total amounts larger than the average order amount.

You could display the names of all the customers who placed an order today by using a join operator or a subquery, but a subquery is not required. This could be accomplished simply by joining the CUSTOMER and CURR_ORDER tables using the CUSTOMER_ID column and including `WHERE order_date = sysdate` to restrict the rows to only those customers placing orders today.

To determine the number of orders placed this year by the customer with CUSTOMER_ID value 30450, you could query the CURR_ORDER table restricting the rows using a WHERE clause and include the COUNT aggregate function in the select list.

To determine the average credit limit of all the customers who opened an account this year, you could join the CUSTOMER and CURR_ORDER tables and restrict the rows using a WHERE clause. Then, use the AVG aggregate function in the select list to calculate

the average credit limit for each group of rows.

Subqueries are often used in a WHERE clause of a SQL statement to return values for an unknown conditional value. The inner query executes first and returns the results to the outer query for use in the outer query's WHERE clause.

Item: 15 (Ref:Cert-1Z0-071.7.8.5)

Examine the structure of the employee table.

EMPLOYEE

EMPLOYEE_ID	NUMBER	NOT NULL, Primary Key
EMP_LNAME	VARCHAR2(25)	
EMP_FNAME	VARCHAR2(25)	
DEPT_ID	NUMBER	Foreign key to DEPT_ID column of DEPARTMENT table
JOB_ID	NUMBER	Foreign key to JOB_ID column of JOB table
MGR_ID	NUMBER	References EMPLOYEE_ID column
SALARY	NUMBER(9,2)	
HIRE_DATE	DATE	
DOB	DATE	

You want to generate a list of employees are in department 30, have been promoted from clerk to associate by querying the employee and employee_hist tables. The employee_hist table has the same structure as the employee table. The job_id value for clerks is 1 and the job_id value for associates is 6.

Which query should you use?

○
```
SELECT employee_id, emp_lname, emp_fname, dept_id
FROM employee
WHERE (employee_id, dept_id) IN
(SELECT employee_id, dept_id
FROM employee_hist
WHERE dept_id = 30 AND job_id = 1)
AND job_id = 6;
```

○
```
SELECT employee_id, emp_lname, emp_fname, dept_id
FROM employee
WHERE (employee_id) IN
(SELECT employee_id
FROM employee_hist
WHERE dept_id = 30 AND job_id = 1);
```

○
```
SELECT employee_id, emp_lname, emp_fname, dept_id
FROM employee
WHERE (employee_id, dept_id) =
(SELECT employee_id, dept_id
FROM employee_hist
WHERE dept_id = 30 AND job_id = 6);
```

○
```
SELECT employee_id, emp_lname, emp_fname, dept_id
FROM employee
WHERE (employee_id, dept_id) IN
(SELECT employee_id, dept_id
FROM employee
WHERE dept_id = 30)
AND job_id = 6;
```

○
```
SELECT employee_id, emp_lname, emp_fname, dept_id
FROM employee_hist
WHERE (employee_id, dept_id) =
(SELECT employee_id, dept_id
FROM employee_hist WHERE dept_id = 30
AND job_id = 1)
AND job_id = 6;
```

Answer:

```
SELECT employee_id, emp_lname, emp_fname, dept_id
FROM employee
WHERE (employee_id, dept_id) IN
(SELECT employee_id, dept_id
FROM employee_hist
WHERE dept_id = 30 AND job_id = 1)
AND job_id = 6;
```

Explanation:

You should use the following query:

```
SELECT employee_id, emp_lname, emp_fname, dept_id
FROM employee
WHERE (employee_id, dept_id) IN
(SELECT employee_id, dept_id
FROM employee_hist
WHERE dept_id = 30 AND job_id = 1)
AND job_id = 6;
```

A multi-column subquery is used to retrieve the employee IDs and department IDs of employees who are clerks (`job_id = 1`) and who work in department 30 from the `employee_hist` table. The `IN` operator is used to compare the list of employee IDs retrieved from the subquery with the employee IDs in the `employee` table (the outer query). This retrieved list of employees is further qualified with the use of the `AND` operator, eliminating any employees from the list that are not currently associates (`job_id = 6`). The result is a list of employees in department 30 who were promoted from clerk to associate.

The `SELECT` statement that returns a single column in the subquery is incorrect because this statement will return a list of all employees who were previously clerks in department 30.

The `SELECT` statement that includes `WHERE dept_id = 30 AND job_id = 6` as the condition for the inner query is incorrect because this statement returns all employees who at any time have been associates in department 30.

Both of the statements that use the same table in the inner and outer query are incorrect. To produce a report of promotions, both tables must be included in the query. The `employee` table must be queried to check for each employee's current job, and the `employee_hist` table must be queried to determine each employee's previous position.

Item: 16 (Ref:Cert-1Z0-071.7.8.3)

Evaluate this `SELECT` statement:

```
SELECT first_name, last_name
FROM physician
WHERE physician_id NOT IN (SELECT physician_id
FROM physician
WHERE license_no = 17852);
```

Which one of the following `SELECT` statements would achieve the same result?

- ○
```
SELECT first_name, last_name
FROM physician
WHERE physician_id = 17852;
```
- ○
```
SELECT first_name, last_name
FROM physician_id
WHERE license_no <> 17852
AND license_no IS NOT NULL;
```
- ○
```
SELECT first_name, last_name
FROM physician
WHERE physician_id IN (SELECT physician_id
FROM physician
WHERE license_no = 17852);
```
- ○
```
SELECT first_name, last_name
```

```
FROM physician
WHERE physician_id != ALL (SELECT physician_id
FROM physician
WHERE license_no = 17852);
```

Answer:

```
SELECT first_name, last_name
FROM physician
WHERE physician_id != ALL (SELECT physician_id
FROM physician
WHERE license_no = 17852);
```

Explanation:

The following SELECT statement will return the same result as the given SELECT statement:

```
SELECT first_name, last_name
FROM physician
WHERE physician_id != ALL (SELECT physician_id
FROM physician
WHERE license_no = 17852);
```

In the scenario, the given SELECT statement uses the NOT IN operator to display all physicians who do not have a license number of 17852. The same results can be achieved using the != and ALL operators. If null values are likely to be returned by the inner query, you should not use either of these operators. If one of the values returned by the inner query is null, the entire query will not return any rows, because all the conditions that compare a NULL value yield a null result.

The SELECT statement that includes WHERE physician_id = 17852 as the query condition is incorrect because it will return only the physician that has an identifier of 17852, and this is not what you desired.

The SELECT statement that includes the PHYSICIAN_ID table in the FROM clause is incorrect. PHYSICIAN_ID is a column in the PHYSICIAN table, not a table itself.

The SELECT statement that uses a subquery with the IN operator returns the opposite of what you needed. The inner query returns the identifier for the physician with a license number of 17852, and then the outer query returns this physician's name. You needed all the physicians who did not have a license number of 17852. Therefore, this option is incorrect.

Item: 17 (Ref:Cert-1Z0-071.7.8.6)

You need to create a report to display the names of customers with a credit limit greater than the average credit limit of all customers.

Which SELECT statement should you use?

○
```
SELECT last_name, first_name
FROM customer
WHERE credit_limit > AVG(credit_limit);
```

○
```
SELECT last_name, first_name, AVG(credit_limit)
FROM customer
GROUP BY AVG(credit_limit);
```

○
```
SELECT last_name, first_name, AVG(credit_limit)
FROM customer
GROUP BY AVG(credit_limit)
HAVING credit_limit > AVG(credit_limit);
```

○
```
SELECT last_name, first_name
FROM customer
WHERE credit_limit > (SELECT AVG(credit_limit)

FROM customer);
```

○
```
SELECT last_name, first_name
FROM customer
WHERE credit_limit = (SELECT AVG(credit_limit)
```

```
FROM customer);
```

Answer:

```
SELECT last_name, first_name
FROM customer
WHERE credit_limit > (SELECT AVG(credit_limit)

FROM customer);
```

Explanation:

You should use the following SELECT statement:

```
SELECT last_name, first_name
FROM customer
WHERE credit_limit > (SELECT AVG(credit_limit)
FROM customer);
```

To return the names of all customers with a credit limit greater than the average credit limit of all customers, you must use the statement that uses a subquery and compares the credit limit to the subquery values using the greater than operator (>). In this scenario, the inner query returns the average credit limit of all customers. The outer query takes this average credit limit value and uses this value to display all the customers who have a credit limit greater than this amount.

The statement that includes `WHERE credit_limit > AVG(credit_limit)` for the query condition is incorrect. Aggregate, or group, functions cannot be used in a `WHERE` clause.

Neither of the statements that group the result by `AVG(credit_limit)` is correct because group functions are not allowed in a `GROUP BY` clause.

The statement that includes a subquery and compares the credit limit to the subquery values using the equality operator (=) will return only those customers who have a credit limit equal to the average credit limit of all customers, and this is not what you desired.

Item: 18 (Ref:Cert-1Z0-071.7.3.1)

You create a script that contains the following query:

```
SELECT Fname, Lname, Salary, DepartmentID
FROM Employees
WHERE Salary = (SELECT MAX (Salary)
          FROM Employees);
```

What type of query is it?

- ◯ Correlated subquery
- ◯ Single row subquery
- ◯ Multiple row subquery
- ◯ Multiple column subquery

Answer:

Single row subquery

Explanation:

The query is an example of a single row subquery. A single row subquery returns a single row in the subquery. In this example, the `WHERE Salary = (SELECT MAX (Salary) FROM Employees)` clause will return a single record of the employee that has the highest salary.

All other answers are incorrect because all other options return more than a single record.

A multiple row subquery by its nature will return multiple rows in the sub query.

A correlated subquery uses the EXISTS operator to check for the existence of data rows that satisfy specified criteria.

A multiple-column subquery returns a multiple columns to the outer query using the outer query's FROM, WHERE, or HAVING clause.

Item: 19 (Ref:Cert-1Z0-071.7.2.2)

Examine the structures of the PLAYER and TEAM tables:

```
PLAYER
-------------
PLAYER_ID NUMBER PK
LAST_NAME VARCHAR2(30)
FIRST_NAME VARCHAR2(25)
TEAM_ID NUMBER
MGR_ID NUMBER
SIGNING_BONUS NUMBER(9,2)

TEAM
---------
TEAM_ID NUMBER
TEAM_NAME VARCHAR2(30)
```

Which situation would require a subquery to return the desired result?

- ◯ a list of all players who are also managers
- ◯ a list of all teams that have more than 11 players
- ◯ a list of all players, including their signing bonus amounts and their manager names
- ◯ a list of all players who have a larger signing bonus than their manager
- ◯ a list of all players who received a signing bonus that was lower than the average bonus

Answer:

a list of all players who received a signing bonus that was lower than the average bonus

---

Explanation:

With the given tables, a subquery would be required to produce a list of all players who received a signing bonus that was lower than the average. To produce a list of players who received a signing bonus that was lower than the average, you could use:

```
SELECT last_name, first_name
FROM player
WHERE signing_bonus < (SELECT AVG(signing_bonus)
FROM player);
```

To produce a list of all players that are also managers, you could use a self join to join the PLAYER table to itself.

To produce a list of all teams with more than 11 players, you could query the PLAYER table grouping the records by TEAM_ID and using the COUNT function to count the distinct values of PLAYER_ID. Then, after grouping the data you could use a HAVING clause to return only those teams having more than 11 players.

To produce a list of all players, including their manager names and signing bonuses, you could use a self join to join the PLAYER table to itself. This would allow you to display each player's manager.

To produce a list of all players who have a larger signing bonus than their manager, you would join the PLAYER table to itself using a self join and then compare the signing bonuses for the player and the manager using a WHERE clause.

Subqueries are often used in a WHERE clause of a SQL statement to return values for an unknown conditional value. The inner query executes first and returns the results to the outer query for use in the outer query's WHERE clause.

Item: 20 (Ref:Cert-1Z0-071.7.8.4)

The employee table contains these columns:

```
EMPLOYEE_ID NUMBER NOT NULL
EMP_LNAME VARCHAR2(20) NOT NULL
EMP_FNAME VARCHAR2(10) NOT NULL
DEPT_ID NUMBER
SALARY NUMBER(9,2)
```

A user needs to retrieve information on employees who have the same department ID and salary as an employee ID that the user will enter. You want the query results to include employees who do not have a salary, but not the employee that the user entered.

Which statement will return the desired result?

○
```
SELECT *
FROM employee
WHERE (department, salary) NOT IN

(SELECT department, salary)
FROM employee
WHERE employee_id = &1);
```

○
```
SELECT *
FROM employee
WHERE (dept_id, salary) IN
(SELECT dept_id, NVL(salary, 0)
FROM employee
WHERE employee_id = &1);
```

○
```
SELECT *
FROM employee
WHERE (dept_id, NVL(salary, 0)) IN

(SELECT dept_id, NVL(salary, 0)
FROM employee
WHERE employee_id = &&1)
AND employee_id <> &&1;
```

○
```
SELECT *
FROM employee
WHERE (dept_id, salary) IN
(SELECT dept_id, salary)
FROM employee
WHERE employee_id = &1
AND salary IS NULL);
```

Answer:

```
SELECT *
FROM employee
WHERE (dept_id, NVL(salary, 0)) IN

(SELECT dept_id, NVL(salary, 0)
FROM employee
WHERE employee_id = &&1)
AND employee_id <> &&1;
```

Explanation:

The following query will retrieve the desired result:

```
SELECT *
FROM employee
WHERE (dept_id, NVL(salary, 0)) IN (SELECT dept_id, NVL(salary, 0)
FROM employee
WHERE employee_id = &&1)
AND employee_id <> &&1;
```

When this statement executes, the inner query is processed first. The inner query returns the `dept_id` and `salary` values for the `employee_id` entered. These values are passed to the outer query, which produces a list of employees having the same department and salary. If a `NULL` value is returned in a subquery, the entire query will return no rows. To ensure that the subquery does not return a `NULL` value for the `salary` column, the `NVL` function is used. Because a value of zero is returned from the subquery if the `salary` value is `NULL`, the query result will include employees that do not have a salary. In addition, the `employee_id <> &&1` condition in the outer query `WHERE` clause will exclude the employee entered from the list.

The `SELECT` statement that references the `department` column is invalid because this is not a valid column name in the `employee` table.

The `SELECT` statement that is similar to the correct statement but includes only one condition in the outer query `WHERE` clause is incorrect. Although this query will return the correct list of employees, it will also return the employee that was entered, and this is not what you desired.

The `SELECT` statement that uses `salary IS NULL` in the inner query `WHERE` clause is incorrect. This inner query will return a `NULL` value, and when a subquery returns a `NULL` value, the entire result is null. Therefore, this query will return no rows.

Item: 21 (Ref:Cert-1Z0-071.7.5.1)

Consider the following `UPDATE` statement:

```
UPDATE books
SET price=price+100, royalty=royalty*1.25, (book_type, publication)=(SELECT category, publication
FROM authors WHERE author_rank=1)
WHERE author_id= (SELECT author_id FROM authors WHERE author_rank=1);
```

Which of the following statements are TRUE about the `UPDATE` statement? (Choose all that apply.)

- ☐ `price` is increased by 100 for all the books.
- ☐ `royalty` is increased by 25% for all the books by the top author.
- ☐ `book_type` is updated for all the books.
- ☐ `publication` is updated for books by the top author.

Answer:

`royalty` is increased by 25% for all the books by the top author.
`publication` is updated for books by the top author.

Explanation:

The correct answer is that royalty is increased by 25% for all the books by the top author and `publication` is updated for books by the top author. The `WHERE` clause in the `UPDATE` statement modifies the values of `price` and `royalty` for the top author. The `author_id` in the `books` table and the `author_id` in the `authors` table for the top author should match in order for `price` and `royalty` to be updated.

The `UPDATE` statement uses a subquery in the `SET` clause wherein `book_type` and `publication` are retrieved from the `authors` table. The subquery returns the values for `book_type` and `publication` only for the top author.

The value of `price` is not increased by 100 for all the books because the `WHERE` clause in the statement is true for only the top author, and `price` is increased for the books by the top author.

The `book_type` is not updated for all the books because the `WHERE` clause in the statement is true for the top author, and hence `book_type` is updated only for the top author.

Item: 22 (Ref:Cert-1Z0-071.7.3.2)

You must produce a report that lists all products ordered in the year 2015 by customers in Atlanta and the invoice numbers for those orders.

Which of the following queries can you use to produce the required report?

○
```
SELECT p.ProductName, s.InvoiceNo
FROM Products p JOIN Sales s ON p.ProductID = s.ProductID
WHERE s.InvoiceNo EXISTS
(SELECT InvoiceNo FROM Invoices
WHERE ShipCity = 'Atlanta' AND EXTRACT(year FROM OrderDate) = 2015)
```

○
```
SELECT p.ProductName, s.InvoiceNo
FROM Sales s JOIN Products p ON s.ProductID = p.ProductID
JOIN Invoices i ON i.InvoiceNo = s.InvoiceNo
WHERE i.ShipCity = 'Atlanta' AND EXTRACT(year FROM OrderDate) = 2015
GROUP BY p.ProductName
```

○
```
SELECT p.ProductName, s.InvoiceNo
FROM Products p JOIN Sales s ON p.ProductID = s.ProductID
WHERE s.InvoiceNo = i.InvoiceNo
AND i.ShipCity = 'Atlanta' AND EXTRACT(year FROM OrderDate) = 2015
```

○
```
SELECT p.ProductName, s.InvoiceNo
FROM Products p JOIN Sales s ON p.ProductID = s.ProductID
WHERE s.InvoiceNo IN
(SELECT InvoiceNo FROM Invoices
WHERE ShipCity = 'Atlanta' AND EXTRACT(year FROM OrderDate) = 2015)
```

Answer:

```
SELECT p.ProductName, s.InvoiceNo
FROM Products p JOIN Sales s ON p.ProductID = s.ProductID
WHERE s.InvoiceNo IN
(SELECT InvoiceNo FROM Invoices
WHERE ShipCity = 'Atlanta' AND EXTRACT(year FROM OrderDate) = 2015)
```

Explanation:

The following query returns two columns:

```
SELECT p.ProductName, s.InvoiceNo
FROM Products p JOIN Sales s ON p.ProductID = s.ProductID
JOIN Invoices i ON s.InvoiceNo = i.InvoiceNo
WHERE i.ShipCity = 'Atlanta' AND EXTRACT(year FROM OrderDate) = 2015
```

The first column contains a list of the products that were sent to Atlanta in 2015. The second column contains the invoice numbers for the corresponding orders that generated those shipments.

The correct query joins the `Products`, `Sales`, and `Invoices` tables. The same result will be produced by using a subquery instead of the second join. The subquery returns the numbers of those invoices that were issued in 2015 and that have a value of `Atlanta` in the `ShipCity` column. The `IN` operator is the equivalent of `=ANY`, which returns true if the value in the `InvoiceNo` column of the `Sales` table in the `WHERE` clause of the outer query matches any of the values returned by the subquery. This is exactly what a join with the `Invoices` table on the `InvoiceNo` column does.

The query that includes `WHERE s.InvoiceNo EXISTS` is invalid because no column name, constant, or expression can precede the `EXISTS` operator.

The query that includes `GROUP BY p.ProductName` is invalid because it includes the `GROUP BY` clause and because the `InvoiceNo` column in the `SELECT` list is not included either in the `GROUP BY` clause or in an aggregate function.

The query that includes `WHERE s.InvoiceNo = i.InvoiceNo`
`AND i.ShipCity = 'Atlanta' AND EXTRACT(year FROM OrderDate) = 2015` is invalid. This query is missing the `Invoices` table, and no alias `i` has been defined. However, the alias `i` appears in the `WHERE` clause and is used to qualify the

`InvoiceNo`, `ShipCity`, and `OrderDate` columns.