

Item: 1 (Ref:Cert-1Z0-071.5.3.3)

Examine the structure of the product table.

PRODUCT

PRODUCT_ID	NUMBER	NOT NULL, Primary Key
PRODUCT_NAME	VARCHAR2 (25)	
SUPPLIER_ID	NUMBER	Foreign key to SUPPLIER_ID of the SUPPLIER table
LIST_PRICE	NUMBER (7,2)	
COST	NUMBER (7,2)	
QTY_IN_STOCK	NUMBER	
QTY_ON_ORDER	NUMBER	
REORDER_LEVEL	NUMBER	
REORDER_QTY	NUMBER	

Evaluate this SQL statement:

```
SELECT supplier_id, AVG(cost)
FROM product
WHERE AVG(list_price) > 60.00
GROUP BY supplier_id
ORDER BY AVG(cost) DESC;
```

Which clause will cause an error?

- SELECT
- WHERE
- GROUP BY
- ORDER BY

Answer:

WHERE

Explanation:

The WHERE clause will cause an error when this SELECT statement is executed. Groups can only be restricted with a HAVING clause. Including a group (or aggregate) function in a WHERE clause is invalid.

All of the other clauses are correctly specified and do not generate errors.

Item: 2 (Ref:Cert-1Z0-071.5.2.1)

Examine the data from the po_detail table.

PO_DETAIL

PO_NUM	PO_LINE_ID	PRODUCT_ID	QUANTITY	UNIT_PRICE
10052	1	1	100	10.30
10052	2	2	100	10.00
10054	1	1	50	72.10
10054	2	1	10	10.00
10054	3	3	10	10.00

You query the po_detail table and a value of 5 is returned.

Which SQL statement did you execute?

- SELECT SUM(quantity)
FROM po_detail;
- SELECT AVG(unit_price)
FROM po_detail;
- SELECT COUNT(AVG(unit_price))
FROM po_detail;
- SELECT COUNT(*)
FROM po_detail;
- SELECT COUNT(DISTINCT product_id)
FROM po_detail;
- SELECT COUNT(po_num, po_line_id)
FROM po_detail;

Answer:

```
SELECT COUNT(*)
FROM po_detail;
```

Explanation:

You executed the following statement:

```
SELECT COUNT(*)
FROM po_detail;
```

The COUNT group function returns the number of rows in a table when COUNT(*) is used without a WHERE clause. The po_detail table contains five rows, so the SELECT statement returns a value of 5.

The SQL statement that includes SUM(quantity) in the select list is incorrect. This statement will return 270, which is the result of adding all values of the QUANTITY column.

The SQL statement that includes AVG(unit_price) in the select list is also incorrect. This statement will return 22.48, or the average of all five values of UNIT_PRICE.

The SQL statement that uses both the COUNT and AVG functions will return an error. Nested functions are allowed and are evaluated from the innermost function to the outermost function. However, if group functions are nested, the statement must contain a GROUP BY clause.

The SQL statement that uses the COUNT function with the DISTINCT keyword will return a value of 3. When the DISTINCT keyword is used with the COUNT function it will return the number of distinct non-null values of the given expression, which in this case is the number of distinct non-null product_id values.

The SQL statement that includes COUNT(po_num, po_line_id) in the select list is invalid and will result in an error. The COUNT function accepts a single column or expression as its argument.

Item: 3 (Ref:Cert-1Z0-071.5.3.1)

Which two statements about the evaluation of clauses in a `SELECT` statement are true? (Choose two.)

- The Oracle Server will evaluate a `HAVING` clause before a `WHERE` clause.
- The Oracle Server will evaluate a `WHERE` clause before a `GROUP BY` clause.
- The Oracle Server will evaluate a `GROUP BY` clause before a `HAVING` clause.
- The Oracle Server will evaluate an `ORDER BY` clause before a `WHERE` clause.
- The Oracle Server will evaluate an `ORDER BY` clause before a `HAVING` clause.

Answer:

The Oracle Server will evaluate a `WHERE` clause before a `GROUP BY` clause.

The Oracle Server will evaluate a `GROUP BY` clause before a `HAVING` clause.

Explanation:

The Oracle Server will evaluate a `WHERE` clause before a `GROUP BY` clause. The Oracle Server will evaluate a `GROUP BY` clause before a `HAVING` clause. The order of evaluation of clauses is:

1. `WHERE` clause
2. `GROUP BY` clause
3. `HAVING` clause
4. `ORDER BY` clause

The `WHERE` clause establishes the candidate rows. From these rows, the Oracle Server identifies the groups in the `GROUP BY` clause. The `HAVING` clause further restricts the result groups. The data is then ordered based on the values in the `ORDER BY` clause.

The statements indicating that the `HAVING` clause will be evaluated before the `WHERE` clause, the `ORDER BY` clause will be evaluated before the `WHERE` clause, or the `ORDER BY` clause will be evaluated before the `HAVING` clause are incorrect.

Item: 4 (Ref:Cert-1Z0-071.5.3.2)

Examine the structures of the `employee` and `department` tables:

`employee`

```
-----
EMP_ID NUMBER NOT NULL PK
NAME VARCHAR(30) NOT NULL
FNAME VARCHAR(25) NOT NULL
DEPT_NO NUMBER
TITLE VARCHAR2(25)
```

`department`

```
-----
DEPT_ID NUMBER NOT NULL PK
DEPT_NAME VARCHAR2(25)
```

You need to produce a list of departments, including the department name, which have more than three administrative assistants.

Which SELECT statement will produce the desired result?

- SELECT dept_name
FROM employee JOIN department
ON employee.dept_id = department.dept_id
WHERE UPPER(title) = 'ADMINISTRATIVE ASSISTANT'
GROUP BY dept_name
HAVING emp_id > 3;
- SELECT dept_name
FROM employee
GROUP BY dept_no
HAVING LOWER(title) = 'administrative assistant' AND COUNT(*) > 3;
- SELECT dept_name
FROM employee NATURAL JOIN department
WHERE LOWER(title) = 'administrative assistant'
GROUP BY dept_name
HAVING COUNT(emp_id) > 3;
- SELECT dept_name
FROM employee e JOIN department d
ON (e.dept_no = d.dept_id)
WHERE LOWER(title) = 'administrative assistant'
AND COUNT(*) > 3;
- SELECT d.dept_name
FROM employee e JOIN department d
ON (e.dept_no = d.dept_id)
WHERE LOWER(title) = 'administrative assistant'
GROUP BY dept_name
HAVING COUNT(emp_id) > 3;
- SELECT d.dept_name
FROM e.employee JOIN d.department
ON (e.dept_no = d.dept_id)
WHERE LOWER(title) = 'administrative assistant'
GROUP BY dept_name
HAVING COUNT(emp_id) > 3;

Answer:

```
SELECT d.dept_name
FROM employee e JOIN department d
ON (e.dept_no = d.dept_id)
WHERE LOWER(title) = 'administrative assistant'
GROUP BY dept_name
HAVING COUNT(emp_id) > 3;
```

Explanation:

The following SELECT statement will produce the desired result:

```
SELECT d.dept_name
FROM employee e JOIN department d
ON (e.dept_no = d.dept_id)
WHERE LOWER(title) = 'administrative assistant'
GROUP BY dept_name
HAVING COUNT(emp_id) > 3;
```

To produce a list of departments having more than three administrative

Item: 5 (Ref:Cert-1Z0-071.5.2.3)

Examine the structure of the product table:

PRODUCT

PRODUCT_ID	NUMBER	NOT NULL, Primary Key
PRODUCT_NAME	VARCHAR2 (25)	
SUPPLIER_ID	NUMBER	Foreign key to SUPPLIER_ID of the SUPPLIER table
LIST_PRICE	NUMBER (7,2)	
COST	NUMBER (7,2)	
QTY_IN_STOCK	NUMBER	
QTY_ON_ORDER	NUMBER	
REORDER_LEVEL	NUMBER	
REORDER_QTY	NUMBER	

Which SELECT statement displays the number of items for which the list_price is greater than \$400.00?

- SELECT SUM(*)
FROM product
WHERE list_price > 400;
- SELECT COUNT(*)
FROM product
ORDER BY list_price;
- SELECT COUNT(*)
FROM product
WHERE list_price > 400;
- SELECT SUM(*)
FROM product
GROUP BY list_price > 400;

Answer:

```
SELECT COUNT(*)
FROM product
WHERE list_price > 400;
```

Explanation:

The following SELECT statement displays the number of items whose list_price is greater than \$400.00:

```
SELECT COUNT(*)
FROM product
WHERE list_price > 400;
```

COUNT(*) returns the number of rows in a table or in a particular group or rows in a table. Because no GROUP BY clause is provided in this statement, all rows in the table that meet the WHERE clause criteria are counted.

Both SELECT statements that use the SUM group function are incorrect. The SUM group function returns the sum of a group of values, which is not what you desired. Additionally, SUM(*) is invalid syntax and will generate an error.

The SELECT statement that uses the COUNT group function but does not include a WHERE clause is also incorrect. To restrict the rows counted to only those with a list price greater than \$400.00, a WHERE clause would need to be included.

Item: 6 (Ref:Cert-1Z0-071.5.3.4)

Evaluate this SQL statement:

```

SELECT manufacturer_id, COUNT(*), order_date
FROM inventory
WHERE price > 5.00
GROUP BY order_date, manufacturer_id
HAVING COUNT(*) > 10
ORDER BY order_date DESC;

```

Which clause specifies which rows will be returned from the inventory table?

- SELECT manufacturer_id, COUNT(*), order_date
- WHERE price > 5.00
- GROUP BY order_date, manufacturer_id
- ORDER BY order_date DESC
- HAVING COUNT(*) > 10

Answer:

WHERE price > 5.00

Explanation:

The WHERE clause uses a condition to qualify or restrict the query results to only rows meeting the condition of PRICE > 5.00 in the inventory table.

All of the other options are incorrect because none of these clauses specifies which rows will be returned from the inventory table. The SELECT clause is used to restrict the columns, or expressions returned from the table. The GROUP BY clause is used to divide query result rows into smaller groups. The HAVING clause is used to further restrict which groups will be returned. The ORDER BY clause is used to sort the rows returned.

Item: 7 (Ref:Cert-1Z0-071.5.2.2)

Examine the data in the WORKORDER table.

WORKORDER

WO_ID	CUST_ID	REQUIRED_DT	COMPL_DT	AMOUNT
1	1	04-DEC-2011	02-DEC-2011	520.32
2	1	02-JAN-2012		
3	2	17-JAN-2012		
4	2	20-JAN-2012	05-JAN-2012	274.11
6	3	14-JAN-2012	13-JAN-2012	400.00
7	3	04-FEB-2012		
8	4	01-FEB-2012		
9	5		14-JAN-2012	

The WORKORDER table contains these columns:

```

WO_ID NUMBER PK
CUST_ID NUMBER
REQUIRED_DT DATE
COMPL_DT DATE
AMOUNT NUMBER(7,2)

```

Which statement regarding the use of aggregate functions on the `WORKORDER` table is TRUE?

- Using the `AVG` aggregate function with any column in the table is allowed.
- Using the `AVG` aggregate function on the `AMOUNT` column ignores null values.
- Using the `MIN` aggregate function on the `COMPL_DT` column will return a null value.
- Using the `SUM` aggregate function with the `AMOUNT` column is allowed in any portion of a `SELECT` statement.
- Using the `SUM` aggregate function on the `AMOUNT` column will result in erroneous results because the column contains null values.
- Grouping on the `REQUIRED_DT` and `COMPL_DT` columns is not allowed.

Answer:

Using the `AVG` aggregate function on the `AMOUNT` column ignores null values.

Explanation:

All group functions, except specifically the `COUNT(*)` function, ignore null values. Therefore, it is correct to state that using the `AVG` aggregate function on the `AMOUNT` column ignores null values.

Grouping is allowed on any column, so the option stating that grouping on the `REQUIRED_DT` and `COMPL_DT` columns is not allowed is incorrect.

The option stating you can use the `AVG` function with any column in the table is incorrect because you can only use the `AVG`, `SUM`, `STDDEV`, and `VARIANCE` functions with numeric data types.

All group functions ignore null values. Therefore, the `MIN` aggregate function would not return a null value, and the option stating that it does is incorrect.

The option that states that the `SUM` function is allowed in any portion of a `SELECT` statement is incorrect. Aggregate functions cannot be used in `WHERE` clauses. `WHERE` clauses restrict the rows before the grouping occurs. `HAVING` clauses restrict the rows after the grouping occurs.

Using the `SUM` function on the `AMOUNT` column will not result in erroneous results due to null values. Aggregate functions ignore null values.

If you need to substitute another value, for example zero, for a null value, you can use the `NVL`, `NVL2`, or `COALESCE` functions.

Item: 8 (Ref:Cert-1Z0-071.5.1.1)

The `EMPLOYEE` table contains these columns:

```
EMP_ID NUMBER(9)
FNAME VARCHAR2(25)
LNAME VARCHAR(30)
SALARY NUMBER(7,2)
BONUS NUMBER(5,2)
DEPT_ID NUMBER(9)
```

You need to calculate the average bonus for all the employees in each department. The average should be calculated based on all the rows in the table, even if some employees do not receive a bonus.

Which group function should you use to calculate this value?

- `AVG`
- `SUM`

- MAX
- MEAN
- COUNT
- AVERAGE

Answer:

AVG

Explanation:

To calculate the average bonus, you should use the `AVG` function. The `AVG` group function can be used to calculate the average value for a group of values.

When using `AVG` and other group functions, null values are ignored and not included in the group calculation. In this scenario, you wanted to include null values, so you should also use the `NVL` function to force the `AVG` function to include null values. In this scenario, you would use this `SELECT` statement to return the desired results:

```
SELECT AVG(NVL(bonus, 0))  
FROM employee  
GROUP BY dept_id;
```

The `NVL` single-row function is used to convert a null to an actual value and can be used on any data type including `VARCHAR2` columns. The syntax for the `NVL` function is:

```
NVL(expression1, expression2)
```

Although `SUM`, `MAX`, and `COUNT` are valid group functions, none of these functions will calculate an average for a group of values as required in the scenario. You would use the `SUM` group function to total a group of values, ignoring null values. You would use the `MAX` group function to return the greatest value in a group of values, ignoring null values. You would use the `COUNT` group function to return the number of rows in a group while ignoring null values. The `COUNT(*)` clause will count all the selected rows, including duplicates and rows with nulls.

`MEAN` and `AVERAGE` are not valid group functions. Therefore, these options are incorrect.