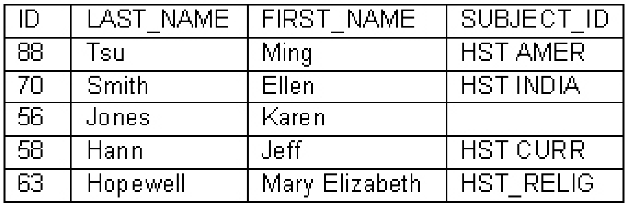
Item: 1 (Ref:1Z0-061.3.1.3)

Examine the data in the TEACHER table.

TEACHER



ID	LAST_NAME	FIRST_NAME	SUBJECT_ID
88	Tsu	Ming	HST AMER
70	Smith	Ellen	HST INDIA
56	Jones	Karen	
58	Hann	Jeff	HST CURR
63	Hopewell	Mary Elizabeth	HST_RELIG

Evaluate this SQL statement:

```
SELECT last_name||', '||first_name
FROM teacher
WHERE subject_id != NULL
ORDER BY last_name;
```

Which value is displayed FIRST when this query is executed?

○ `Tsu, Ming`

○ `Hann, Jeff`

○ `Smith, Ellen`

○ No value is displayed.

○ `Jones, Karen`

○ `Hopewell, Mary Elizabeth`

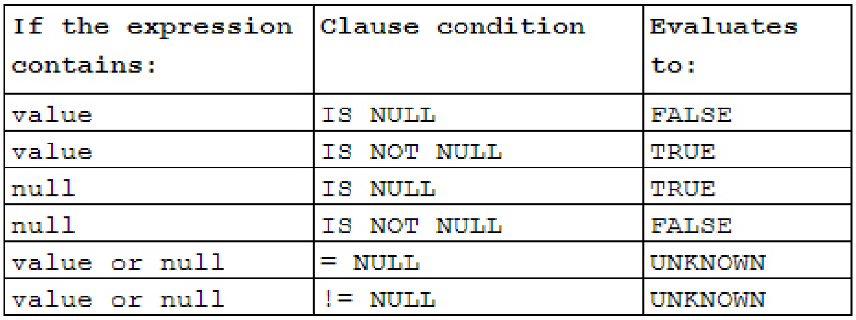
Answer:

No value is displayed.

Explanation:

No value is displayed. Because a `NULL` value cannot be compared to any value, comparison operators such as equal (=), greater than (>), less than (<), and not equal to (!= or <>) will not produce the desired result. The `NOT` keyword should be used with the `IS NULL` operator (`IS NOT NULL`) to test for `NULL` values, or the absence of data.

The following table summarizes how an expression is evaluated with different WHERE clause condition operators:



If the expression contains:	Clause condition	Evaluates to:
value	IS NULL	FALSE
value	IS NOT NULL	TRUE
null	IS NULL	TRUE
null	IS NOT NULL	FALSE
value or null	= NULL	UNKNOWN
value or null	!= NULL	UNKNOWN

Item: 2 (Ref:1Z0-061.3.1.9)

The `product` table contains these columns:

```
PRODUCT_ID NUMBER(9)
DESCRIPTION VARCHAR2(20)
COST NUMBER(5,2)
MANUFACTURER_ID VARCHAR2(10)
```

You want to display product costs with these desired results:

- The cost displayed for each product is twenty percent greater than the current price stored in the table
- The product's manufacturer ID is 25001, 25020, or 25050.
- Twenty percent of the product's original cost is less than $4.00.

Which statement should you use?

○
```
SELECT description, cost * .20
FROM product
WHERE cost * .20 < 4.00
AND manufacturer_id BETWEEN '25001' AND '25050';
```

○
```
SELECT description, cost * 1.20
FROM product
WHERE cost * .20 < 4
AND manufacturer_id = ('25001', '25020', '25050');
```

○
```
SELECT description, cost * 1.20
FROM product
WHERE cost * .20 < 4.00
AND manufacturer_id IN ('25001', '25020', '25050');
```

○
```
SELECT description, cost * 1.20
FROM product
WHERE cost * .20 < 4.00
AND manufacturer_id ANY('25001', '25020', '25050');
```

Answer:

```
SELECT description, cost * 1.20
FROM product
WHERE cost * .20 < 4.00
AND manufacturer_id IN ('25001', '25020', '25050');
```

Explanation:

You should use the following statement:

```
SELECT description, cost * 1.20
FROM product
WHERE cost * .20 < 4.00
AND manufacturer_id IN ('25001', '25020', '25050');
```

The desired result that the cost displayed be increased by 20 percent is achieved by multiplying `cost` by `1.20`. By multiplying the cost by 120 percent, 20 percent of the cost is added to the existing cost.

The desired result that the product's `manufacturer_id` be equal to 25001, 25050, or 25050 is achieved by using the `IN` operator. The `IN` operator is used to match a column or expression's value with any of the values specified in a list. The effect is the same as using the `OR` operator. The `NOT` keyword may precede the `IN` operator. The correct syntax using the `IN` operator is:

```
column IN(value1 [,value2] ...)
```

The desired result that the cost, when multiplied by 20 percent, be less than $4.00 is achieved by using the less than (<) operator.

You should not use the statement containing the `BETWEEN` operator. The statement containing the `BETWEEN` operator (`BETWEEN '25001' AND '25050'`) achieves only one of the desired results. The statement successfully executes, but returns more rows than desired. The first desired result, which specifies that the cost displayed be increased by 20 percent, is not achieved when multiplying the price by .20. The second desired result that specifies an item manufacturer ID of 25001, 25050, or 25050 is not achieved because the `BETWEEN` operator returns all rows that have a `manufacturer_id` value between `25001` and `25050`.

You should not use the statement containing the equality (=) operator. The statement containing the equality (=) operator (`manufacturer_id = ('25001', '25020', '25050')`) fails. The = operator must be compared to only one value. In this scenario, it is compared to three values.

You should not use the statement containing the `ANY` operator. Using the `ANY` operator (`ANY('25001', '25020', '25050')`) without an equality, inequality, or less than or greater than operator preceding it will cause the entire statement to fail.

Item: 3 (Ref:1Z0-061.3.2.4)

The LINE_ITEM table contains these columns:

```
LINE_ITEM_ID NUMBER(9) Primary Key
ORDER_ID NUMBER(9)
PRODUCT_ID VARCHAR2(9)
QUANTITY NUMBER(5)
```

Evaluate this SQL statement:

```
SELECT quantity, product_id
FROM line_item
ORDER BY quantity, product_id;
```

Which statement is true concerning the results of executing this statement?

○ The results are sorted numerically only.

○ The results are sorted alphabetically only.

○ The results are sorted numerically and then alphabetically.

○ The results are sorted alphabetically and then numerically.

Answer:

The results are sorted numerically and then alphabetically.

Explanation:

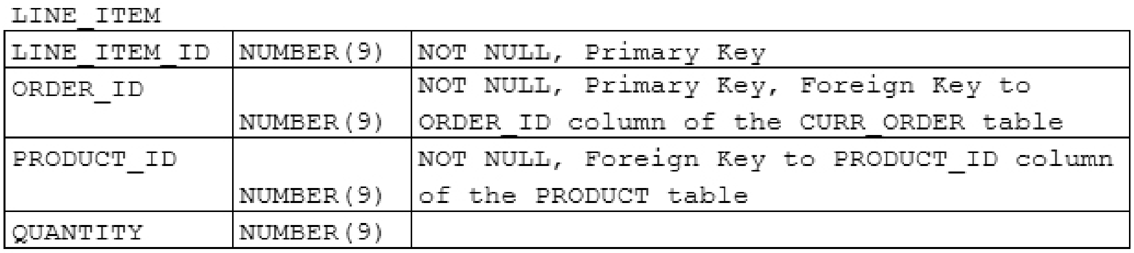
The results are sorted numerically and then alphabetically. The ORDER BY clause sorts rows in descending or ascending order. Because the quantity column is listed first in the ORDER BY clause, that sort will be performed first. The quantity column has a numeric datatype, so the rows will first be sorted numerically. The product_id column has a VARCHAR2 data type, so the rows will be sorted alphabetically for any duplicate quantity values.

The results are not sorted numerically only or alphabetically only because the columns listed in the ORDER BY clause are of different data types, numeric and character.

The sort is not alphabetic and then numeric because the quantity column, which is a NUMBER data type, is listed in the ORDER BY clause before the product_id column, which is a VARCHAR2 data type.

Item: 4 (Ref:1Z0-061.3.1.4)

Examine the structure of the LINE_ITEM table.



LINE_ITEM

LINE_ITEM_ID	NUMBER(9)	NOT NULL, Primary Key
ORDER_ID	NUMBER(9)	NOT NULL, Primary Key, Foreign Key to ORDER_ID column of the CURR_ORDER table
PRODUCT_ID	NUMBER(9)	NOT NULL, Foreign Key to PRODUCT_ID column of the PRODUCT table
QUANTITY	NUMBER(9)	

You want to display order ID numbers, product ID numbers, and the quantities ordered for each of the products ordered with these desired results:

- The volume of the item ordered must be 50 or greater.
- The displayed results must be sorted from lowest to the highest by order ID number, and then by product ID number.
- The items must belong to order numbers ranging from 1800 to 1900.

Evaluate this SQL statement:

```
SELECT order_id, product_id, quantity
FROM line_item
WHERE quantity >= 50
AND order_id IN(1800, 1900)
ORDER BY order_id, product_id;
```

Which statement about using this query as the proposed solution is true?

○ One of the desired results is achieved.

○ Two of the desired results are achieved.

○ All of the desired results are achieved.

○ The statement generates an error.

Answer:

Two of the desired results are achieved.

Explanation:

With the SELECT statement given in this scenario, two of the desired results are achieved. The first desired result, which specifies that the volume of the item ordered must be 50 or greater, is achieved with the clause WHERE quantity >= 50. The second desired result, which specifies that the displayed results be sorted from lowest to highest by the order ID number and then by the product ID number, is achieved using the clause ORDER BY order_id, product_id.

When more than one column is listed in an ORDER BY clause, the results are sorted on the first column. If duplicate values are returned on the first column, then a sort on the next column in the ORDER BY clause occurs.

The third desired result is not achieved. The given SELECT statement uses a condition of order_id IN(1800, 1900) as one of the WHERE clause conditions, which only returns rows that have an order_id value of 1800 or 1900. To achieve the third desired result, you should use the BETWEEN operator to test for a value in the specified range of 1800 through 1900, inclusive. The BETWEEN operator restricts the query results to be based on a column value being greater than or equal to a specified low value and less than or equal to a specified high value.

In this scenario, the following query would achieve all of the desired results:

1Z0-061: Restricting and Sorting

```
SELECT order_id, product_id, quantity
FROM line_item
WHERE quantity >= 50
AND order_id BETWEEN 1800 AND 1900
ORDER BY order_id, product_id;
```

Item: 5 (Ref:1Z0-061.3.3.1)

Evaluate this SELECT statement:

```
SELECT order_num, &order_date
FROM &&ordertbl
WHERE order_date = '&order_date';
```

Which statement regarding the execution of this statement is true?

○ The user will be prompted for the table name each time the statement is executed in a session.

○ The user will be prompted for the table name only the first time the statement is executed in a session.

○ The user will be prompted for all values in the select list each time the statement is executed in a session.

○ An error will occur when executing this statement because substitution variables are not allowed in a WHERE clause.

○ An error will occur when executing this statement because substitution variables must be unique within a SELECT statement.

Answer:

The user will be prompted for the table name only the first time the statement is executed in a session.

Explanation:

When executing this statement, the user will be prompted for the table name only the first time the statement is executed within a session. Substitution variables specified with the double ampersand (&&) are reusable. The user is prompted only the first time the statement executes. The variable then remains available until the session ends or an UNDEFINE command is issued for the variable. Substitution variables specified with the single ampersand (&) will prompt the user each time the statement is executed.

The user will not be prompted for the table name each time the statement is executed because the substitution variable used in the FROM clause contains a double ampersand (&&).

The user will be prompted for one of the values in the SELECT list each time the statement is executed. The reason for this is that only one of the two variables in the SELECT list uses a substitution variable. The other value in the SELECT list uses the column name defined in the table. Thus, its value will be retrieved from the table, and not by prompting the user to interactively enter a value.

Substitution variables are allowed in any part of a SELECT statement. Therefore, the option stating that an error will occur because substitution variables are not allowed in a WHERE clause is incorrect.

Substitution variables can be used more than once within a single SELECT statement. In this statement &order_date is referenced twice. This is acceptable and does not generate an error. Therefore, the option stating that an error occurs because substitution variables must be unique within a SELECT statement is incorrect.

Item: 6 (Ref:1Z0-061.3.2.2)

You want to query employee information and display the results sorted by the employee's department, then by their salaries from highest to lowest. When multiple employees within the same department share a last name, they must be displayed in alphabetical order by first name.

Which `ORDER BY` clause should you use in your query?

○ `ORDER BY department_id, salary, last_name, first_name`

○ `ORDER BY department_id, salary ASC, last_name, first_name`

○ `ORDER BY department_id, salary DESC, last_name, first_name`

○ `ORDER BY department_id, salary DESC, first_name ||' '|| last_name ASC`

Answer:

`ORDER BY department_id, salary DESC, last_name, first_name`

Explanation:

You should use the following `ORDER BY` clause:

```
ORDER BY department_id, salary DESC, last_name, first_name
```

Because the default sort order when using an `ORDER BY` clause is ascending (lowest to highest for numeric data, earliest to latest for date data, and alphabetically for character data), only the `salary` column needs to use the `DESC` (the reverse of ascending order) keyword to display salaries from the highest to the lowest.

The `ORDER BY` clause containing no keywords and the `ORDER BY` clause containing `salary ASC` do not return the desired results because the `salary` data is displayed in ascending order rather than descending order.

The `ORDER BY` clause containing the concatenated columns executes successfully, but does not return the desired results of placing the first names in alphabetical order when the last names are identical.

Item: 7 (Ref:1Z0-061.3.1.8)

For which task would you use a WHERE clause in a SELECT statement?

○ to display only rows with a product_id value of 7382

○ to designate the order table location

○ to display only unique product_id values

○ to restrict the rows returned by a GROUP BY clause

Answer:

to display only rows with a product_id value of 7382

Explanation:

You would use a WHERE clause in a SELECT statement to display only rows with a product_id value of 7382. A WHERE clause uses a condition to qualify or restrict query results, such as to compare product_id values to 7382 using WHERE product_id = 7382. A WHERE clause directly follows the FROM clause and contains one or more conditions that must be met for a row to be returned in the query results.

Logical conditions may be created using the NOT, AND, and OR operators. Comparison conditions may also be created using these operators: =, >, <, >=, <=, <>, !=, [NOT] BETWEEN...AND... , [NOT] IN, ANY, ALL, SOME, EXISTS, LIKE, and IS [NOT] NULL.

You would not use a WHERE clause in a SELECT statement to designate the order table location. The storage location of a table can be established when creating a table or altered after table creation. This is performed using the CREATE TABLE or ALTER TABLE statements.

You would not use a WHERE clause in a SELECT statement to display only unique product_id values. To display only the unique product_id values, use the DISTINCT keyword in a SELECT clause.

You would not use a WHERE clause in a SELECT statement to restrict the rows returned by a GROUP BY clause. To restrict the rows returned by a GROUP BY clause, use the HAVING clause. The condition or conditions specified in the HAVING clause are evaluated after applying any WHERE clause condition and performing the grouping.

Item: 8 (Ref:1Z0-061.3.2.7)

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
CATEGORY_ID	NUMBER	
QTY_PER_UNIT	NUMBER	
LIST_PRICE	NUMBER(5,2)	
COST	NUMBER(5,2)	

You want to display the product identification numbers of all products with 500 or more units available for immediate sale. You want the product identification numbers displayed numerically by supplier identification number, then by product identification number from lowest to highest.

Which statement should you use to achieve the required results?

○
```
SELECT product_id
FROM product
WHERE qty_per_unit >= 500
ORDER BY supplier_id, product_id;
```

○
```
SELECT product_id
FROM product
WHERE qty_per_unit >= 500
SORT BY supplier_id, product_id;
```

○
```
SELECT product_id
FROM product
WHERE qty_per_unit >= 500
ORDER BY supplier_id, product_id DESC;
```

○
```
SELECT product_id
FROM product
WHERE qty_per_unit > 500
SORT BY supplier_id, product_id;
```

Answer:

```
SELECT product_id
FROM product
WHERE qty_per_unit >= 500
ORDER BY supplier_id, product_id;
```

Explanation:

You should use the following statement to achieve the desired results:

```
SELECT product_id
FROM product
WHERE qty_per_unit >= 500
ORDER BY supplier_id, product_id;
```

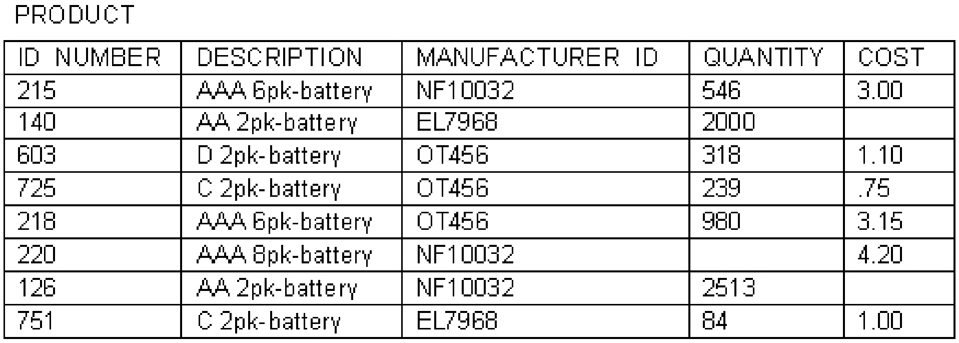
With this statement, all identification numbers of products that have a quantity greater than or equal to 500 are displayed because the greater than or equal to (>=) comparison operator is used. The `ORDER BY` clause must contain the supplier_id and product_id columns as requested. The product identification numbers are displayed by supplier identification number in ascending order, and then within supplier identification number by product identification number, ordered from lowest to highest. The default sort order when using an ORDER BY clause is ascending (lowest to highest for numeric data, earliest to latest for date data, and alphabetically for character data). Because this is the default sort order, no sort order keyword, ASC or DESC, is required in the ORDER BY clause.

You should not use either of the statements that contain a SORT BY clause. Both of these statements will fail to execute because SORT BY is not a valid SELECT statement clause. If SORT BY were replaced with ORDER BY, one of the statements would execute returning the desired results. The other would execute and return undesired results.

You should not use the statement that contains the DESC keyword. This statement executes successfully, but does not return the desired results. The DESC keyword of the ORDER BY clause orders the values in ascending order by supplier identification number, but then within supplier identification number the values are sorted from the highest to the lowest by product identification number. In this scenario, you wanted the rows to be sorted in ascending order by product identification number.

Item: 9 (Ref:1Z0-061.3.2.3)

Examine the data in the `product` table.



PRODUCT

ID NUMBER	DESCRIPTION	MANUFACTURER ID	QUANTITY	COST
215	AAA 6pk-battery	NF10032	546	3.00
140	AA 2pk-battery	EL7968	2000	
603	D 2pk-battery	OT456	318	1.10
725	C 2pk-battery	OT456	239	.75
218	AAA 6pk-battery	OT456	980	3.15
220	AAA 8pk-battery	NF10032		4.20
126	AA 2pk-battery	NF10032	2513	
751	C 2pk-battery	EL7968	84	1.00

You execute the following query:

```
SELECT description, quantity, cost
FROM product
WHERE manufacturer_id LIKE 'NF10032'
AND NVL(cost, 0) < 5.00
ORDER BY quantity DESC, cost;
```

Which result will the query provide?

○
```
DESCRIPTION QUANTITY     COST
----------- ------------ ---------

AA          2pk-battery  2513
AAA         6pk-battery  546 3
```

○
```
DESCRIPTION QUANTITY     COST
----------- ------------ ---------
AAA         8pk-battery  4.2
AA          2pk-battery  2513
AAA         6pk-battery  546 3
```

○
```
DESCRIPTION QUANTITY     COST
----------- ------------ ---------

AAA         6pk-battery  546 3
AAA         8pk-battery  4.2
AA          2pk-battery  2513
```

○
```
DESCRIPTION QUANTITY     COST
----------- ------------ ---------
AA          2pk-battery  2513
AAA         6pk-battery  546 3
AAA         8pk-battery  4.2
```

Answer:

```
DESCRIPTION QUANTITY     COST
----------- ------------ ---------
AAA         8pk-battery  4.2
AA          2pk-battery  2513
AAA         6pk-battery  546 3
```

Explanation:

The query will provide the following result:

```
DESCRIPTION QUANTITY     COST
----------- ------------ ---------
AAA         8pk-battery  4.2
AA          2pk-battery  2513
AAA         6pk-battery  546 3
```

When a query is performed using an ascending sort on a column with null values, the null values are displayed last. When the default sort order of ascending (lowest to highest for numeric data, earliest to latest for date data, and alphabetically order for character data) is overridden using the `DESC` keyword (the reverse of ascending), the resulting display is reversed with the null value(s) being displayed first. After any null values are displayed, the remaining values are displayed from the highest value to the lowest, as shown in this scenario.

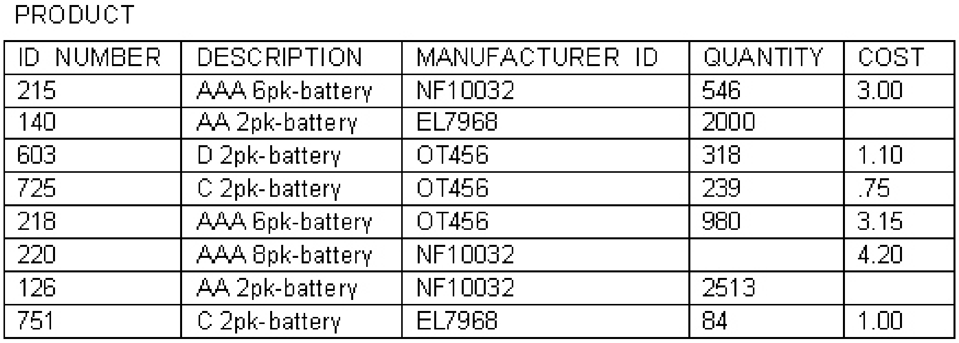
The result that is missing the record with the null `QUANTITY` value is incorrect because any null `QUANTITY` values should be displayed.

The result that is displaying the `QUANTITY` value of `546` first is incorrect because no sort order is used.

The result that is displaying the highest `QUANTITY` value first is incorrect because of the descending sort order that will place any null values first.

Item: 10 (Ref:1Z0-061.3.2.1)

Examine the data in the `product` table.



PRODUCT

ID NUMBER	DESCRIPTION	MANUFACTURER ID	QUANTITY	COST
215	AAA 6pk-battery	NF10032	546	3.00
140	AA 2pk-battery	EL7968	2000	
603	D 2pk-battery	OT456	318	1.10
725	C 2pk-battery	OT456	239	.75
218	AAA 6pk-battery	OT456	980	3.15
220	AAA 8pk-battery	NF10032		4.20
126	AA 2pk-battery	NF10032	2513	
751	C 2pk-battery	EL7968	84	1.00

Evaluate this `SELECT` statement:

```
SELECT description, cost
FROM product
ORDER BY cost, quantity;
```

Which statements are true? (Choose all that apply.)

□ The `product_id` value for the first record displayed is `220`.

□ The `product_id` values for the last two rows displayed are `140` and `126`.

□ The `description` value for the first two rows displayed is `C 2pk-battery`.

□ The `description` value for the first two rows displayed is `AA 2pk-battery`.

□ No row with a `product_id` of `220` is displayed.

Answer:

The `product_id` values for the last two rows displayed are `140` and `126`.

The `description` value for the first two rows displayed is `C 2pk-battery`.

Explanation:

The following statements are true in this scenario:

- The `product_id` values for the last two rows displayed are `140` and `126`.
- The `description` value for the first two rows displayed is `C 2pk-battery`.

The `ORDER BY` clause specifies the order of the data retrieved by the query. The default sort order is ascending. When data is sorted in ascending sequence, null values are displayed last. In this scenario, rows with a null value in the `cost` column are displayed last. The `product_id`s associated with these rows are `140` and `126`. The products with the lowest cost determine which descriptions are displayed first. The lowest `cost` values are `.75` and `1.00`. The first two `description` values, which are associated with these two `cost` values, are both `C 2pk-battery`. Because the lowest `cost` values are `.75` and `1.00`, and there is only one of each of these values, a further sort on the `quantity` column is not necessary.

The `product_id` of the first record displayed is `725`. This `product_id` is associated to the lowest `cost` value of `.75`.

The first two `description` values are `C 2pk-battery`. If null values were displayed first, instead of last , the `description` value of `AA 2pk-battery` would be displayed for the first two rows.

A row with the `product_id` of `220` is displayed when this query is executed.

Item: 11 (Ref:1Z0-061.3.1.1)

The `account` table contains these columns:

```
ACCOUNT_ID NUMBER(12)
NEW_BALANCE NUMBER(7,2)
PREV_BALANCE NUMBER(7,2)
FINANCE_CHARGE NUMBER(7,2)
```

You need to create a single `SELECT` statement to accomplish these requirements:

- Display accounts that have a new balance that is less than the previous balance.
- Display accounts that have a finance charge that is less than $25.00.
- Display accounts that have no finance charge.

Evaluate this statement:

```
SELECT account_id
FROM account
WHERE new_balance < prev_balance
AND NVL(finance_charge, 0) < 25;
```

How many of the three requirements will this `SELECT` statement accomplish?

○ all of the requirements

○ one of the requirements

○ two of the requirements

○ none of the requirements

Answer:

all of the requirements

Explanation:

The given `SELECT` statement will accomplish all of the requirements. The first desired result, to display accounts with a new balance less than the previous balance, is achieved with the `WHERE` clause condition `new_balance < prev_balance`. The second and third desired results, to display accounts with a finance charge less than $25.00 and accounts without a finance charge, are achieved with the `WHERE` clause condition `NVL(finance_charge, 0) < 25`. 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)
```

If `expression1` is null, `NVL` returns `expression2`. If `expression1` is not null, `NVL` returns `expression1`. The `expression1` and `expression2` arguments can be of any datatype. When the expression datatypes differ, Oracle converts `expression2` to the datatype of `expression1` before the two expressions are compared.

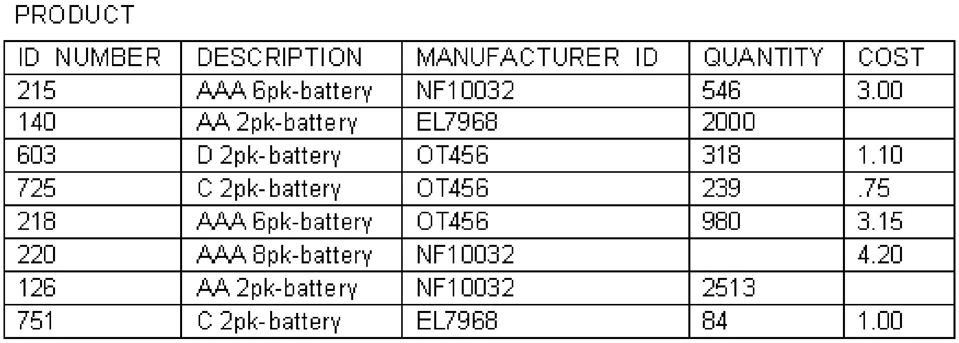
This query may also be used to display all of the desired results:

```
SELECT account_id
FROM account
WHERE new_balance < prev_balance
AND finance_charge < 25
OR finance_charge IS NULL;
```

All other options are incorrect because the given `SELECT` statement will accomplish all three requirements.

Item: 12 (Ref:1Z0-061.3.2.8)

Examine the data in the `product` table.



PRODUCT

ID NUMBER	DESCRIPTION	MANUFACTURER ID	QUANTITY	COST
215	AAA 6pk-battery	NF10032	546	3.00
140	AA 2pk-battery	EL7968	2000	
603	D 2pk-battery	OT456	318	1.10
725	C 2pk-battery	OT456	239	.75
218	AAA 6pk-battery	OT456	980	3.15
220	AAA 8pk-battery	NF10032		4.20
126	AA 2pk-battery	NF10032	2513	
751	C 2pk-battery	EL7968	84	1.00

You query the `product` table with this SQL statement:

```
SELECT description
FROM product
ORDER BY manufacturer_id, quantity ASC;
```

What is the `description` value of the first row displayed?

○ C 2pk-battery

○ D 2pk-battery

○ AA 2pk-battery

○ AAA 6pk-battery

Answer:

C 2pk-battery

Explanation:

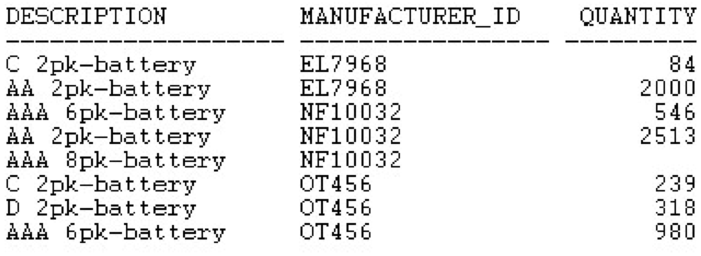
The `description` value of the first row displayed is `C 2pk-battery`. When using an `ORDER BY` clause to sort the rows returned from a query, the default sort order is ascending. An ascending sort order displays values from lowest to highest for numeric data, from earliest to latest for date data, and alphabetically for character data. Because ascending is the default sort order, it is not necessary to append the `ASC` keyword to the `ORDER BY` clause.

Because there is an `ORDER BY` clause on the `manufacturer_id` column, the `manufacturer_id` column values will be used to order the results, even though only the `description` column is displayed. The first `manufacturer_id` value alphabetically is `EL7968`. Of the two rows with a `manufacturer_id` value of `EL7968` in the `product` table, the `description` value of the row with the lowest `quantity` will be displayed first because there is also an `ORDER BY` on `quantity`. This row has a `description` value of `C 2pk-battery`.

The `description` value of the first row displayed would be `D 2pk-battery` if the `ORDER BY` clause were on the `description` column with a descending sort order.

The `description` value of the first row displayed would be `AA 2pk-battery` if the `ORDER BY` clause were on the `description` column with the default ascending sort order.

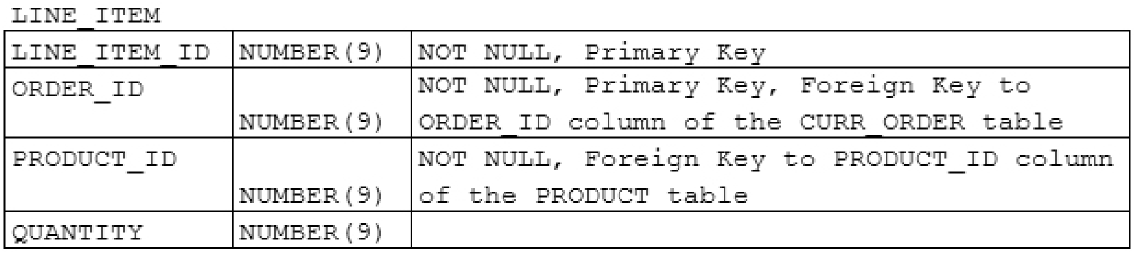
The `description` value of the first row displayed would be `AAA 6pk-battery` if a descending `ORDER BY` clause were used on both the `manufacturer_id` and the `quantity` columns.



```
DESCRIPTION          MANUFACTURER_ID     QUANTITY
-------------------- ------------------ ---------
C 2pk-battery        EL7968                    84
AA 2pk-battery       EL7968                  2000
AAA 6pk-battery      NF10032                  546
AA 2pk-battery       NF10032                 2513
AAA 8pk-battery      NF10032
C 2pk-battery        OT456                    239
D 2pk-battery        OT456                    318
AAA 6pk-battery      OT456                    980
```

Item: 13 (Ref:1Z0-061.3.2.5)

Examine the structure of the `line_item` table.



LINE_ITEM

LINE_ITEM_ID	NUMBER(9)	NOT NULL, Primary Key
ORDER_ID	NUMBER(9)	NOT NULL, Primary Key, Foreign Key to ORDER_ID column of the CURR_ORDER table
PRODUCT_ID	NUMBER(9)	NOT NULL, Foreign Key to PRODUCT_ID column of the PRODUCT table
QUANTITY	NUMBER(9)	

You must display the order number, line item number, product identification number, and quantity of each item where the quantity ranges from 10 through 100. The order numbers must be in the range of 1500 through 1575. The results must be sorted by order number from lowest to highest, and then further sorted by quantity from highest to lowest.

Which statement should you use to display the desired results?

○
```
SELECT order_id, line_item_id, product_id, quantity
FROM line_item
WHERE quantity BETWEEN 9 AND 101
AND order_id BETWEEN 1500 AND 1575
ORDER BY order_id DESC, quantity DESC;
```

○
```
SELECT order_id, line_item_id, product_id, quantity
FROM line_item
WHERE (quantity > 10 AND quantity < 100)
AND order_id BETWEEN 1500 AND 1575
ORDER BY order_id ASC, quantity;
```

○
```
SELECT order_id, line_item_id, product_id, quantity
FROM line_item
WHERE (quantity > 9 OR quantity < 101)
AND order_id BETWEEN 1500 AND 1575
ORDER BY order_id, quantity;
```

○
```
SELECT order_id, line_item_id, product_id, quantity
FROM line_item
WHERE quantity BETWEEN 10 AND 100
AND order_id BETWEEN 1500 AND 1575
ORDER BY order_id, quantity DESC;
```

Answer:

```
SELECT order_id, line_item_id, product_id, quantity
FROM line_item
WHERE quantity BETWEEN 10 AND 100
AND order_id BETWEEN 1500 AND 1575
ORDER BY order_id, quantity DESC;
```

Explanation:

You should use the following statement to display the desired results:

```
SELECT order_id, line_item_id, product_id, quantity
FROM line_item
WHERE quantity BETWEEN 10 AND 100
AND order_id BETWEEN 1500 AND 1575
ORDER BY order_id, quantity DESC;
```

In this statement, the first BETWEEN operator is used to retrieve rows with quantity values of 10 and 100 and all values in between. The second BETWEEN operator is used to retrieve rows with order number values of 1500 and 1575 and all values between. Because the default sort order is ascending (from lowest to highest for numeric data, from earliest to latest for date data, and alphabetically for character data), no keyword is necessary following the order_id column in the ORDER BY clause. To display the quantity values in descending order (from highest to lowest) for each group of order numbers, the DESC keyword is used.

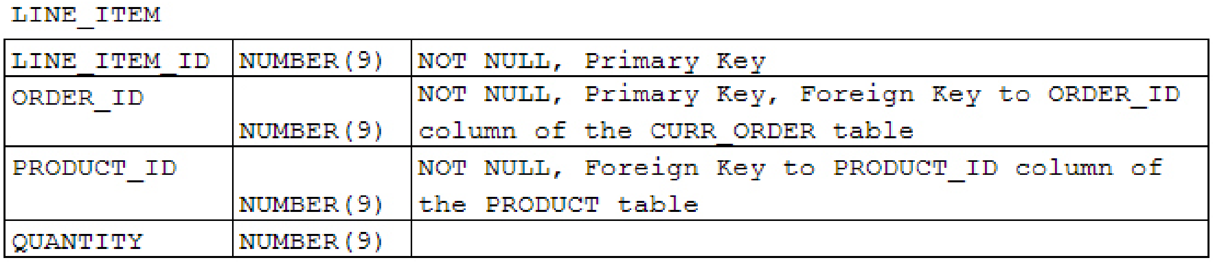
The SELECT statement containing BETWEEN 9 AND 101 will not return the desired results because rows with a quantity value between 10 and 100 will not be returned. The sort order of DESC on the order_id column will also cause undesired results to display.

The SELECT statement containing (quantity > 10 AND quantity < 100) will not return the desired results because rows with a quantity value of 10 and 100 will not be included in the result. A sort order of descending should also be placed on the quantity column in the ORDER BY clause to return the desired results.

The SELECT statement containing (quantity > 9 OR quantity < 101) will not return the desired results. Any row with a quantity value greater than 9 or less than 101 will be returned. A sort order of descending should also be placed on the quantity column in the ORDER BY clause to return the desired results.

Item: 14 (Ref:1Z0-061.3.1.5)

Examine the structure of the `LINE_ITEM` table.



LINE_ITEM

LINE_ITEM_ID	NUMBER(9)	NOT NULL, Primary Key
ORDER_ID	NUMBER(9)	NOT NULL, Primary Key, Foreign Key to ORDER_ID column of the CURR_ORDER table
PRODUCT_ID	NUMBER(9)	NOT NULL, Foreign Key to PRODUCT_ID column of the PRODUCT table
QUANTITY	NUMBER(9)	

You attempt to query the database with this SQL statement:

```
SELECT order_id "Order Number", product_id "Product", quantity "Amount"
FROM line_item
WHERE "Order Number" = 5570
ORDER BY "Amount";
```

This statement fails when executed. Which action should you take to correct the problem?

○ Specify a sort order of `ASC` or `DESC` in the `ORDER BY` clause.

○ Enclose all of the column aliases in single quotes instead of double quotes.

○ Remove the column alias from the `WHERE` clause and use the column name.

○ Remove the column alias from the `ORDER BY` clause and use the column name.

Answer:

Remove the column alias from the `WHERE` clause and use the column name.

Explanation:

To correct the problem, you should remove the column alias from the `WHERE` clause and use the column name. A column alias cannot be used in a `WHERE` clause to identify a column. To correct the problem, you should remove the column alias from the `WHERE` clause and use the column name instead.

Explicitly specifying whether the sort order is ascending or descending is certainly permissible, but is never required, and therefore does not cause a syntax error. Also, the use of an alias column header is acceptable in the `ORDER BY` clause. You can use the actual column name, the alias column header, or an integer representing the ordinal value of the sort column from its position in the `SELECT` clause.

An `ORDER BY` clause does not need to contain the `ASC` or `DESC` keywords. The default sort order of an `ORDER BY` clause is `ASC`, from lowest to highest for numeric data, from earliest to latest for date data, and alphabetically for character data. The `DESC` keyword is used when the reverse sort order is desired.

You should not enclose all of the column aliases in single quotes instead of double quotes. Column aliases must be enclosed in double quotes (`"`) if they require initial capitalization or include spaces.

You should not remove the column alias from the `ORDER BY` clause and use the column name. A column alias can be used to identify a column in an `ORDER BY` clause.

Item: 15 (Ref:1Z0-061.3.1.2)

Evaluate this SQL statement:

```
SELECT l.order_id, i.description, l.quantity
WHERE i.id_number = l.product_id
FROM inventory i, line_item l
ORDER BY l.order_id, i.description;
```

This statement fails when executed. Which change should you make to correct the problem?

○ Reorder the clauses in the statement.

○ Remove the table aliases from the FROM clause.

○ Use the table names instead of the table aliases in the ORDER BY clause.

○ Remove the table alias from the ORDER BY clause, and use only the column name.

Answer:

Reorder the clauses in the statement.

Explanation:

To correct the problem, you should reorder the clauses in the statement. For this statement to execute successfully, the FROM clause must be placed before the WHERE clause. The sequence in which the clauses of a SELECT statement should be placed is: SELECT, FROM, WHERE, GROUP BY, HAVING, and ORDER BY. When a SELECT statement clause is out of sequence, the statement will not execute.

You should not remove the table aliases from the FROM clause to correct the problem. All of the SELECT statement clauses are using the defined table aliases correctly.

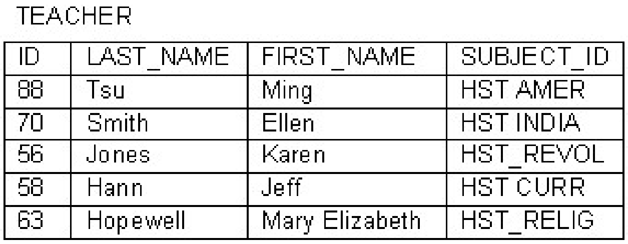
You should not use the table names instead of the table aliases in the ORDER BY clause to correct the problem. The Oracle Server does not require a defined table alias to be used in an ORDER BY clause.

You should not remove the table alias from the ORDER BY clause and use only the column name to correct the problem. The statement will still generate a syntax error since the clauses would continue to be out of order.

A WHERE clause uses a condition to qualify or restrict query results by comparing values. A WHERE clause directly follows the FROM clause and contains a condition(s) that must be met. Logical conditions may be created using the NOT, AND, and OR operators. Comparison conditions may be created using these operators: =, >, <, >=, <=, <>, !=, [NOT] BETWEEN...AND... , [NOT] IN, ANY, ALL, SOME, EXISTS, LIKE, and IS [NOT] NULL.

Item: 16 (Ref:1Z0-061.3.1.6)

Examine the data in the `TEACHER` table.



TEACHER

ID	LAST_NAME	FIRST_NAME	SUBJECT_ID
88	Tsu	Ming	HST AMER
70	Smith	Ellen	HST INDIA
56	Jones	Karen	HST_REVOL
58	Hann	Jeff	HST CURR
63	Hopewell	Mary Elizabeth	HST_RELIG

You want to query the `TEACHER` table and display the following results:

```
Name Subject
------------------------------------- -------------------
Jones, Karen HST_REVOL
Hopewell, Mary Elizabeth HST_RELIG
```

What should you use to query the `TEACHER` table?

○
```
SELECT last_name||', '||first_name "Name", subject_id "Subject"
FROM teacher
WHERE subject_id = 'HST\_R%';
```

○
```
SELECT last_name||', '||first_name "Name", subject_id "Subject"
FROM teacher
WHERE subject_id LIKE 'HST_%';
```

○
```
SELECT last_name||', '||first_name "Name", subject_id "Subject"
FROM teacher
WHERE subject_id LIKE '%HST\_R%' ESC '\';
```

○
```
SELECT last_name||', '||first_name "Name", subject_id "Subject"

FROM teacher
WHERE subject_id LIKE 'HST\_R%' ESCAPE '\';
```

Answer:

```
SELECT last_name||', '||first_name "Name", subject_id "Subject"

FROM teacher
WHERE subject_id LIKE 'HST\_R%' ESCAPE '\';
```

Explanation:

You should use the following query to return the desired results from the `teacher` table:

```
SELECT last_name||', '||first_name "Name", subject_id "Subject"
FROM teacher
WHERE subject_id LIKE 'HST\_R%' ESCAPE '\';
```

When using the `LIKE` operator, you can use the `ESCAPE` option to use the percent (`%`) and underscore (_) characters as literals. Otherwise, they are interpreted as special pattern-matching characters. The `ESCAPE` option is used to identify the escape character. If the escape character appears in the pattern before the characters `%` or _, Oracle interprets the character literally in the pattern, rather than as a special pattern-matching character. The `ESCAPE` option identifies the backslash (`\`) as the escape character in this scenario. In the pattern `HST\_R%`, the escape character precedes the underscore (_). This causes Oracle to interpret the underscore (_) literally.

You should not use the query containing `WHERE subject_id = 'HST\_R%'` as the `WHERE` clause condition. This query executes successfully, but does not return the desired rows. Because the equality operator was used, only rows that had a `subject_id` of `HST\_R%` would be returned. This is because the wildcard characters are only meaningful if the operator is `LIKE` or `NOT LIKE`.

You should not use the query containing `WHERE subject_id LIKE 'HST_%'` as the `WHERE` clause condition. This query executes successfully, but does not return the desired results because all of the rows in the `teacher` table are returned.

You should not use the query containing `WHERE subject_id LIKE '%HST\_R%' ESC '\'` as the `WHERE` clause condition. This query fails because the `ESCAPE` keyword may not be abbreviated.

Item: 17 (Ref:1Z0-061.3.1.7)

Which `SELECT` statement should you use to limit the display of account information to those accounts with a finance charge greater than $75.00?

○
```
SELECT account_id, new_balance, finance_charge
FROM account
WHERE finance_charge > 75.00;
```

○
```
SELECT account_id, new_balance, finance_charge
FROM account
HAVING finance_charge > 75.00;
```

○
```
SELECT account_id, new_balance, finance_charge
FROM account
WHERE finance_charge > 75.00
GROUP BY finance_charge;
```

○
```
SELECT account_id, new_balance, finance_charge
FROM account
GROUP BY finance_charge > 75.00;
```

Answer:

```
SELECT account_id, new_balance, finance_charge
FROM account
WHERE finance_charge > 75.00;
```

Explanation:

You should use the following `SELECT` statement to limit the display of account information to those accounts with a finance charge greater than $75.00:

```
SELECT account_id, new_balance, finance_charge
FROM account
WHERE finance_charge > 75.00;
```

A `WHERE` clause is used to restrict the rows returned by a query, and consists of the `WHERE` keyword followed by a condition. A condition may be composed of column names, expressions, constants, comparison operators, and logical operators. The `WHERE finance_charge > 75.00` clause of the `SELECT` statement limits the display of account information to those accounts with a finance charge greater than 75.00.

You should not use the `SELECT` statement that includes a `HAVING` clause because this statement will fail. A `HAVING` clause is used to further restrict the groups of rows displayed after grouping has occurred with a `GROUP BY` clause. If you include a `HAVING` clause, the query must also contain a `GROUP BY` clause or the statement fails.

The statements containing the improper use of the `GROUP BY` clause fail to execute. In the `GROUP BY price > 75.00` clause, the `GROUP BY` clause is improperly used in the place of a `WHERE` clause. If `GROUP BY` were replaced with `WHERE`, this statement would return the desired results. The statement containing `GROUP BY finance_charge` fails because the `GROUP BY` clause is used without a grouping function in the `SELECT` statement. A `GROUP BY` clause is correctly used to divide query result rows into smaller groups.

Item: 18 (Ref:1Z0-061.3.1.10)

You query the database with this SQL statement:

```
SELECT bonus
FROM salary
WHERE bonus BETWEEN 1 AND 250
OR (bonus IN(190, 500, 600)
AND bonus BETWEEN 250 AND 500);
```

Which `bonus` value could the statement return?

○ 100

○ 260

○ 400

○ 600

Answer:

100

Explanation:

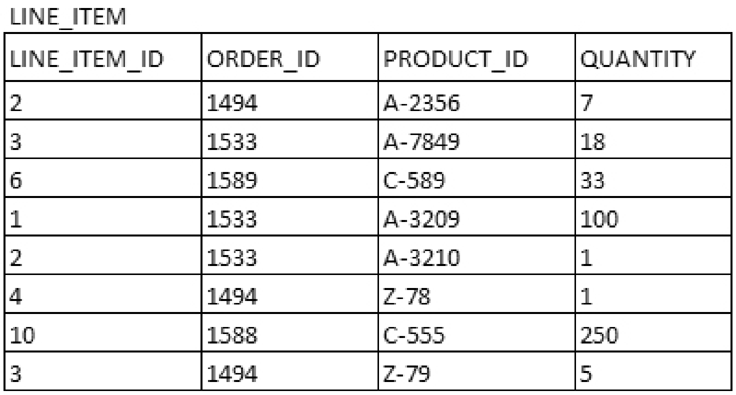
The only listed `bonus` value that this `SELECT` statement could return is 100. The first condition in the `WHERE` clause, `WHERE bonus BETWEEN 1 AND 250`, returns rows with bonus values between 1 and 250 inclusive. In the second and third conditions in the `WHERE` clause, the combination of the two clauses allows only a value of 500 to be returned. The `OR` operator joining the two clauses allows either a value between 1 and 250 or a value of 500 to be returned. The value of 100 is the only value that meets these criteria.

The `BETWEEN` condition has precedence over the `AND` and `OR` conditions. The `AND` condition has precedence over the `OR` condition. Conditions with higher precedence are evaluated first.

All of the other options are incorrect because these values could not be returned by the given statement.

Item: 19 (Ref:1Z0-061.3.2.6)

Examine the data in the `LINE_ITEM` table.



LINE_ITEM

LINE_ITEM_ID	ORDER_ID	PRODUCT_ID	QUANTITY
2	1494	A-2356	7
3	1533	A-7849	18
6	1589	C-589	33
1	1533	A-3209	100
2	1533	A-3210	1
4	1494	Z-78	1
10	1588	C-555	250
3	1494	Z-79	5

Evaluate this SQL statement:

```
SELECT product_id, quantity
FROM line_item
WHERE quantity BETWEEN 5 AND 30
ORDER BY order_id, line_item_id;
```

Which `product_id` value would be displayed last?

○ Z-79

○ C-555

○ A-7849

○ A-2356

Answer:

A-7849

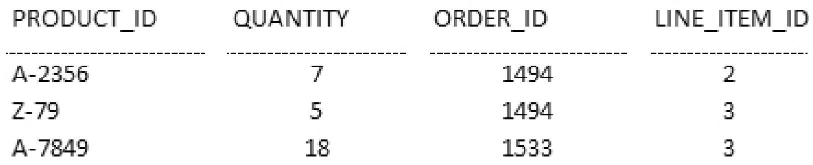
Explanation:

The `product_id` value of `A-7849` would be displayed last. Using the `BETWEEN` operator in the given SQL statement displays all the identification numbers for products that have a quantity of 5, 30, or any quantity between 5 and 30. The product identification numbers are displayed by `order_id` in ascending order, and within `order_id` by `line_item_id` in ascending order. The default sort order when using an `ORDER BY` clause is ascending (lowest to highest for numeric data, earliest to latest for date data, and alphabetically for character data). Because this is the default sort order, no sort order keyword, `ASC` or `DESC`, is required.

The value `Z-79` would be the first `product_id` returned if the `ORDER BY` were on `quantity`.

The value `C-555` would not be returned from this query because the `quantity` associated with it does not fall within the `WHERE` clause criteria.

The value `A-2356` is the first `product_id` value displayed.



```
PRODUCT_ID          QUANTITY          ORDER_ID          LINE_ITEM_ID
-----------------   ---------------   ---------------   ----------------
A-2356                     7               1494                2
Z-79                       5               1494                3
A-7849                    18               1533                3
```