Item: 1 (Ref:Cert-1Z0-071.3.2.1)

Which three functions can be used to manipulate character column values? (Choose three.)

- [ ] RPAD
- [ ] TRUNC
- [ ] ROUND
- [ ] INSTR
- [ ] CONCAT

Answer:

RPAD
INSTR
CONCAT

Explanation:

The following three functions can be used to manipulate character column values:

RPAD
INSTR
CONCAT

The RPAD character function returns a left-justified value with a specified string of characters replicated as many times as necessary to create a string with a length equal to a specified number of character positions (n). The syntax of the RPAD function is:

```
RPAD(column|expression, n, 'string')
```

The INSTR character function is used to find the position of an occurrence of a string of characters within a column value or expression. The Oracle Server begins its character string search at a specified position (m). The Oracle Server will search for the specific occurrence of the string indicated by a specified integer (n). The syntax of the INSTR function is:

```
INSTR(column|expression, 'string' [, m] [, n])
```

The CONCAT character function combines one value (column or expression) to another value (column or expression) and is equivalent to the concatenation operator (||). The syntax of the CONCAT function is:

```
CONCAT(column1|expression1, column2|expression2)
```

TRUNC is used as a date or number function and cannot be used on character column values. The TRUNC date function returns a value with the time portion of the day truncated to the specified format unit for a column value of DATE data type or for a date expression. If no format model ('fmt') is provided, the date is truncated to the nearest day. The syntax of the TRUNC date function is:

```
TRUNC(column|expression [, 'fmt'])
```

The TRUNC number function returns value (a column of number data type or a number expression) truncated to a specified integer representing the number of decimal places (n). The syntax of the TRUNC number function is:

```
TRUNC(column|expression [, n])
```

ROUND is a date and number function and cannot be used on character column values. The ROUND number function is used to round values to a specified decimal place. When an integer representing the number of decimal places to be used (n) is not provided, the number is rounded to zero (0) places. ROUND may be used as either a number or a date function. The syntax of the ROUND number function is:

```
ROUND(column|expression, n)
```

The ROUND date function is used to return a value rounded by a specified format model for a column value of DATE data type or for a date expression. The syntax of the ROUND date function is:

```
ROUND(column|expression [, 'fmt'])
```

Item: 2 (Ref:Cert-1Z0-071.3.1.4)

Which statement concerning SQL functions is TRUE?

- ◯ All date functions return DATE data type values.
- ◯ Character functions can return character or number values.
- ◯ Single-row functions can only be used in SELECT and WHERE clauses.
- ◯ Conversion functions convert a column definition from one data type to another data type.

Answer:

Character functions can return character or number values.

Explanation:

In SQL, it is possible for character functions to return a value that is either a number or a date. For example, the function LENGTH has an input argument of a character data type, and the function returns a number value as an integer. The return value of the function LENGTH('oracle'), in which the input argument is the character string oracle, will return a number, in this case the number 6 since there are six characters in oracle. Another example is the TO_DATE function, which accepts a string (character) input value and returns that same value as a date. The return value of the function TO_DATE('04-Jul-11'), in which the input argument is the character string enclosed in the single ticks ('), will be the date value corresponding to that character string. In this case, that would be July 4, 2011 at midnight.

The two types of character functions are case-manipulation functions and character-manipulation functions. The case-manipulation functions are LOWER, UPPER, and INITCAP. A few examples of character-manipulation functions are CONCAT, SUBSTR, LENGTH, and INSTR.

The option that states all date functions return DATE data type values is incorrect. Although many date functions return DATE data type values, not all do. For example, the MONTHS_BETWEEN function returns a number value.

The option that states single-row functions can only be used in SELECT and WHERE clauses is incorrect. Single-row functions may be used in SELECT, WHERE, and ORDER BY clauses. They can also be nested.

The option that states conversion functions convert a column definition from one data type to another data type is incorrect. Conversion functions convert a value, not a database table column definition, from one data type to another data type.

Item: 3 (Ref:Cert-1Z0-071.3.2.6)

You query the database with this SQL statement:

```
SELECT CONCAT(LOWER(SUBSTR(description, 1, 3)), subject_id) "Subject Description"
FROM subject;
```

In which order are the functions evaluated?

- ◯ CONCAT, LOWER, SUBSTR
- ◯ SUBSTR, LOWER, CONCAT
- ◯ LOWER, SUBSTR, CONCAT
- ◯ All three will be evaluated simultaneously.

Answer:

SUBSTR, LOWER, CONCAT

Explanation:

Nested functions are evaluated from the innermost function to the outermost function. In this scenario, the SUBSTR function will be evaluated first, followed by the LOWER function, with the CONCAT function evaluated last.

The SUBSTR character function returns a portion of a character string, beginning at a specific position (m) that is a specific length (n). A position of zero (0) is treated as a 1. When a substring length is not provided, the Oracle Server returns all of the characters to the end of the string. Null is returned when a substring length is less than 1. The syntax of the SUBSTR function is:

```
SUBSTR(column|expression, m [, n])
```

The LOWER character function converts the value of a column or expression to lower case. The syntax of the LOWER function is:

```
LOWER(column|expression)
```

The CONCAT character function combines one value (column1|expression1) to another value (column2|expression2) and is equivalent to the concatenation operator (||). The syntax of the CONCAT function is:

```
CONCAT(column1|expression1, column2|expression2)
```

All other options stating that a function other than the SUBSTR function is evaluated first or that all three will be evaluated simultaneously are incorrect.

Item: 4 (Ref:Cert-1Z0-071.3.2.5)

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

You query the database and return the value 23.

Which SELECT statement did you use?

- ○
```
SELECT SUBSTR(product_id, 3)
FROM line_item
WHERE line_item_id = 2
AND order_id = 1494;
```
- ○
```
SELECT SUBSTR(product_id, 3, -2)
FROM line_item
WHERE line_item_id = 2
AND order_id = 1494;
```
- ○
```
SELECT SUBSTR(product_id, -3, 2)
FROM line_item
WHERE line_item_id = 2
AND order_id = 1494;
```
- ○
```
SELECT SUBSTR(product_id, 3, 2)
FROM line_item
WHERE line_item_id = 2
AND order_id = 1494;
```

Answer:

```
SELECT SUBSTR(product_id, 3, 2)
FROM line_item
WHERE line_item_id = 2
AND order_id = 1494;
```

Explanation:

You used the following SELECT statement:

```
SELECT SUBSTR(product_id, 3, 2)
FROM line_item
WHERE line_item_id = 2
AND order_id = 1494;
```

In this scenario, the product_id value of A-2356 is returned by the query. The SUBSTR character function returns a portion of a character string, beginning at a specific position (m) that is a specific length (n). A position of 0 is treated as a 1. When a substring length is not provided, Oracle returns all of the characters to the end of the

string. Null is returned when a substring length is less than 1. The syntax of the SUBSTR function is:

```
SUBSTR(column|expression, m [, n])
```

The SUBSTR function position argument of 3 causes the count of characters to start at the third character in the product_id value and count forward. The substring of characters to be displayed will begin with the '2' character. The SUBSTR function length argument of 2 tells the Oracle Server how many characters to count forward, resulting in the value of 23 being displayed.

All of the incorrect options execute successfully, but do not return the given result.

The SELECT statement using SUBSTR(product_id, 3) displays the value 2356.

The SELECT statement using SUBSTR(product_id, 3, -2) returns null values.

The SELECT statement using SUBSTR(product_id, -3, 2) displays the value 35. The SUBSTR function position argument of -3 causes the count of characters to start at the end of the value, counting backwards. The substring of characters to be displayed will begin with the '6' character. The SUBSTR length argument of 2 tells the Oracle Server how many characters to count forward, resulting in the value of 35 being displayed.

Item: 5 (Ref:Cert-1Z0-071.3.2.3)

The PRODUCT table contains these columns:

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

Evaluate these two statements:

Statement 1:
```
SELECT NVL(100 / quantity, 'none')FROM PRODUCT;
```

Statement 2:
```
SELECT NVL(TO_CHAR(quantity), 'none') FROM PRODUCT;
```

Which of the following statements is TRUE?

- ◯ Both statements execute successfully.
- ◯ Statement 1 may fail because the data types are incompatible.
- ◯ Statement 2 causes an error when quantity values are null.
- ◯ Statement 1 executes, but does not display the value 'none' for null values.

Answer:

Statement 1 may fail because the data types are incompatible.

Explanation:

Statement 2 executes successfully, but Statement 1 causes an `ORA-01722: invalid number` error because the data types are incompatible. If the first expression, 100 / quantity, is evaluated and returns a null value, the null value will not be replaced by the character string `none` in the query results because the `quantity` column has a `NUMBER` data type and `none` is a character value. This statement will fail even if there are no null `quantity` values because of the incompatible data types. The solution is to convert the `quantity` column data into character data:

```
SELECT NVL(TO_CHAR(100 / quantity), 'none')
```

<font size="2" fac

Item: 6 (Ref:Cert-1Z0-071.3.1.3)

Which one of the following statements about columns with a data type of `DATE` is true?

- ◯ Dates in Oracle are always stored with a time component. If a user enters a date as part of an `INSERT` statement and no time is specified, the time component will be equal to the time of the `INSERT`.
- ◯ Dates in Oracle are always stored with a century component as a 4-digit year. If the user today enters a date with a value of `45` for the year using a format mask of `RR` for the year, it will be stored as `1945`.
- ◯ Some arithmetic operations with columns that have a data type of `DATE` are possible, but other arithmetic operations on dates are forbidden. For example, you cannot add or subtract 3.14 to a column called `PROCESSING_DATE` if that column is data type `DATE`.
- ◯ It is permissible to subtract one column from another in a table, assuming both columns are data type `DATE`. If the two date columns are storing the same value, the result will be 0. If they are not the same, the result will always be a whole number, but it may be negative.
- ◯ SYSDATE represents the current date and time and has the same precision as any other Oracle column with a data type of `DATE`. If you issued the `SELECT ADD_MONTHS(SYSDATE,12) FROM dual;` command, the result would be the output of the date exactly one year from the day and time you issued the command.

Answer:

`SYSDATE` represents the current date and time and has the same precision as any other Oracle column with a data type of `DATE`. If you issued the `SELECT ADD_MONTHS(SYSDATE,12) FROM dual;` command, the result would be the output of the date exactly one year from the day and time you issued the command.

Explanation:

The value (function) `SYSDATE` represents the current date and time and has the same precision as any other Oracle column with a data type of `DATE`. Adding twelve months to that would give the current date and time exactly one year from the time you issued the `SELECT ADD_MONTHS(SYSDATE,12) FROM dual;` command.

When you insert a value from a column with a data type of `DATE`, and you do not supply a value for the time component of `DATE`, the time is stored as midnight.

If you use the `RR` format mask when storing a 2-digit year, the assumed century will be the 21st if the two digit year is less than or equal to 50. If greater than 50, it assumes the 20th century.

It is permissible to add or subtract a numeric constant, such as 3.25, to a date. Adding to a date is interpreted as

adding that additional number of days to the date. Since .25 is 1/4, and 1/4 of 24 hours is 6 hours, this statement will add (or subtract) 3 days and 6 hours to (from) the value of `processing_date`.

Subtraction of one date minus another is possible. The result represents the number of days (and partial days if there is a decimal result) between the two dates. The answer may be negative depending on the order you subtract, but it also can have a decimal component.

Item: 7 (Ref:Cert-1Z0-071.3.2.2)

You query the database with this SQL statement:

```
SELECT AVG(LENGTH(name)) COLUMN1, SUM(INSTR(ssn,'52',2,2)) COLUMN2 FROM emp2 WHERE
name = INITCAP (name);
```

Review the structure and data of the `emp2` table.

Structure:

NAME	VARCHAR2(25)
SSN	VARCHAR2(9)

Data:

NAME	SSN
Joe	452852452
Mary	444525962
fred	445212525
Tom	492525252
Lucy	172826152

What will be displayed for the output of `COLUMN1` and `COLUMN2`?

- ◯ The value in `COLUMN1` will be `3.5` and the value in `COLUMN2` will be `11`.
- ◯ The value in `COLUMN1` will be `3.6` and the value in `COLUMN2` will be `19`.
- ◯ The value in `COLUMN1` will be `3.5` and the value in `COLUMN2` will be `26`.
- ◯ The value in `COLUMN1` will be `3.6` and the value in `COLUMN2` will be `26`.
- ◯ None of the above will be displayed for the output.

Answer:

The value in `COLUMN1` will be `3.5` and the value in `COLUMN2` will be `11`.

Explanation:

The value in COLUMN1 will be 3.5 and the value in COLUMN2 will be 11. To calculate these values, you should first realize that the row with the name fred should not be included in the calculations because the data for this row does not satisfy the condition of the WHERE clause. The comparison for that row is fred = Fred, which is not true. The lengths for the names of the remaining rows are 3, 4, 3, and 4, respectively. The average of those four values is 3.5.

The value of COLUMN2 is calculated by summing the position number of the second occurrence of "52" in the ssn column, with that search beginning at the second position. Remembering to disregard the fred row, the values of the positions of the second occurrence of 52 are 5, 0, 6, and 0. The sum of those values is 11.

All of the other choices are incorrect since the sum and average of a group of numbers only produces a single result for the sum and a single result for the average. Since those values have already been shown to equal 3.5 for the average and 11 for the sum, any other values must be incorrect.

Item: 8 (Ref:Cert-1Z0-071.3.1.1)

Which of the following is NOT an aggregate function?

- ◯ SUM
- ◯ MAX
- ◯ LENGTH
- ◯ COUNT
- ◯ AVG

Answer:

LENGTH

Explanation:

The LENGTH function is not an aggregate function. Aggregate functions take in multiple values that are part of a set and then return a single value. LENGTH just takes in a single value, namely a string, and returns the number of characters in that string.

SUM is an aggregate function because it takes in a set of values and returns the arithmetic value of the sum when those values are all added together.

MAX is an aggregate function because it takes in a set of values and returns the value from the set that is the greatest.

COUNT is an aggregate function because it takes in a set of values and returns a number representing the number of items in the set.

AVG is an aggregate function because it takes in a set of values and then returns a value representing the sum of all the values in the set divided by the number of values in the set.

Item: 9 (Ref:Cert-1Z0-071.3.2.4)

The `student` table contains these columns:

```
ID NUMBER(9)
LAST_NAME VARCHAR2(25)
FIRST_NAME VARCHAR2(25)
ENROLL_DATE DATE
```

Evaluate these two statements:

Statement 1:
```
SELECT CONCAT(INITCAP(first_name), INITCAP(last_name))
FROM student
WHERE enroll_date < '31-DEC-2007';
```

Statement 2:
```
SELECT INITCAP(first_name) || INITCAP(last_name)
FROM student
WHERE enroll_date < '31-DEC-2007';
```

Which of the following statements is TRUE?

- ◯ The two statements will display the same data.
- ◯ The two statements will not display the same data.
- ◯ One of the statements will fail because it contains a syntax error.
- ◯ The two statements will retrieve the same data, but the display format of the returned values will differ.

Answer:

The two statements will display the same data.

Explanation:

In this scenario, the two statements will return the same data and display the data in the same format, but the column headings will differ. If the `student` table contains a person with a `first_name` value of `JACK` and a `last_name` value of `BROWN`, the name would be displayed by both statements as follows:

```
JackBrown
```

The other options stating that the statements do not display the same data, fail, or retrieve the same data with different display formats of returned values, are all incorrect.

The `CONCAT` character function and the concatenation operator (`||`) are used to concatenate, or combine, one value (`column1|expression1`) to another value (`column2|expression2`). When using the `CONCAT` function, only two values may be joined. When using the concatenation operator, any number of values may be joined. The syntax of the `CONCAT` function is:

```
CONCAT(column1|expression1, column2|expression2)
```

The syntax of the concatenation operator (`||`) is:

```
(column1|expression1)||(column2|expression2)[||(column3|expression3)]...
```

The INITCAP character function converts a mixed case, uppercase, or lowercase value (a column or expression) to a value whose first character is uppercase with the remaining characters in lowercase. The syntax of the INITCAP function is:

```
INITCAP(column|expression)
```

Item: 10 (Ref:Cert-1Z0-071.3.1.2)

Examine the following EMPLOYEE table structure and data:

EMPLOYEE table (structure):

NAME	VARCHAR2(25)
HIREDATE	DATE

EMPLOYEE table (data):

NAME	HIREDATE
George	22-Jul-1989
Aima	15-May-2005
Claus	3-Dec-2000
Johann	
Delvin	29-Feb-2004
Troy	7-Jan-2003
Ann	
Bob	27-Dec-2010
Joshua	15-Mar-1995

From the EMPLOYEE table, you must produce a sorted report that shows each employee's name and the number of years each employee has worked for the company, with the most senior employee first. All employees were continuously employed. Years of employment should be rounded to one decimal place.

A few employees do not yet have a hire date, so their hiredate column in the EMPLOYEE table is NULL. These employees should be listed on the report with zero (0) years of seniority. The column representing the years of seniority should be labeled Seniority, capitalized exactly as shown.

Which of the following SELECT statements will produce the required results?

- ○ `SELECT NAME, HIREDATE, NVL(ROUND((SYSDATE - HIREDATE)/365,1),0) "Seniority" FROM EMPLOYEE ORDER BY hiredate DESC`
- ○ `SELECT NAME, HIREDATE, NVL(ROUND((SYSDATE - HIREDATE)/365,1),0) Seniority FROM EMPLOYEE ORDER BY 3`
- ○ `SELECT NAME, HIREDATE, NVL(ROUND((SYSDATE - HIREDATE)/365,1),0) "Seniority" FROM EMPLOYEE ORDER BY hiredate`
- ○ `SELECT NAME, HIREDATE, NVL(ROUND((SYSDATE - HIREDATE)/365),1),0) "Seniority" FROM EMPLOYEE ORDER BY hiredate`

Answer:

```
SELECT NAME, HIREDATE, NVL(ROUND((SYSDATE - HIREDATE)/365,1),0) "Seniority"
FROM EMPLOYEE ORDER BY hiredate DESC
```

Explanation:

The `SELECT` statement with the clause `ORDER BY hiredate DESC` and the column header `"Seniority"` enclosed in double quotes (") will produce the desired results. In this scenario you want the most senior employes to be listed first.

The `SELECT` statement with the clause `ORDER BY hiredate` will produce the correct values for seniority and the column header will be correct. However, the rows will not be displayed in the correct order.

The `SELECT` statement with the clause `ORDER BY 3` will produce the correct values for seniority in the correct order. However, the column header will be incorrect because it is not enclosed in double quotes. The column header produced by this statement will be `SENIORITY` rather than `Seniority`.

The `SELECT` statement with the clause `NVL(ROUND((SYSDATE HIREDATE)/365),1),0)` will produce an error because of a mismatch in the parentheses. The close parenthesis character `)` should not immediately follow `365`. Note that this `SELECT` statement contains a total of three left parentheses and four right parentheses. This always indicates an error condition due to a mismatch in the number of left and right parentheses.