Item: 1 (Ref:1Z0-061.5.3.1)

The following SQL command was issued at approximately 10:00 AM on the database server machine. The output produced is also displayed below.

```
SQL> SELECT SYSDATE T1, TRUNC(SYSDATE) T2, ROUND(SYSDATE) T3 FROM dual;

T1        T2        T3
--------- --------- ---------
10-MAR-11 10-MAR-11 10-MAR-11
```

Which of the following statements which are TRUE regarding this `SELECT` statement and its corresponding output? (Choose all that apply).

- ☐ The actual value for `T1` also includes the century, the hour, the minute, and the second.
- ☐ The actual values for `T1`, `T2`, and `T3` are all different, even though their output is identical.
- ☐ If you subtracted `T3` from `T2`, the result would be the whole number zero (0).
- ☐ If you added 12 to `T1`, the output displayed for that result would be `10-MAR-12`.
- ☐ If you reissued the same SQL statement 3 hours later, the output displayed for `T1` and `T2` would be the same, but the output for T3 would be `11-MAR-11`.

Answer:

The actual value for `T1` also includes the century, the hour, the minute, and the second.

If you subtracted `T3` from `T2`, the result would be the whole number zero (0).

If you reissued the same SQL statement 3 hours later, the output displayed for `T1` and `T2` would be the same, but the output for T3 would be `11-MAR-11`.

Explanation:

The values `T1`, `T2`, and `T3` are all dates. Therefore, Oracle maintains the necessary details to store in the database any specific second in the date/time range between January 1, 4712 BC and December 31, 9999 AD. That means Oracle maintain the following details for dates:

- the AD or BC indicator
- the century
- the year
- the month
- the day
- the hour
- the minute
- the second

This level of detail was stored into the `DATE` data type column or variable at the time the column or variable was inserted into the table. Since many users do not provide that level of detail when storing a date, Oracle provides several default values. If you inserted 25-Dec-11 into a column called `hiredate`, the day would be 25, the month would be December, and the year would be 11. However, by default the database includes the fact that the AD/BC indicator is AD, the century is the current century, and the hours, minutes, and seconds are all zero. If you stored `SYSDATE` for someone's `hiredate`, the value stored would depend upon when you issued the command,

but it would look similar to March 10, 2011 AD at 9 hours, 32 minutes, and 17 seconds past midnight.

The following is true for each of the values:

- T1 is `SYSDATE`, so it has stored all the detail down to the second of when the user issued the statement.
- T2, which is the `TRUNC` of T1, will store the same value as T1 except where you drop (truncate) the hour/minute/second component of T1, you will change the time component (hours, minutes, second) of the column back to 0/0/0, which is midnight of the same date.
- T3 is the round of T1, so you have to decide whether the original date you are given, which in this case is `SYSDATE`, is closer to midnight of the current day or closer to midnight of the following. If the date falls before noon, rounding `SYSDATE` will set you equal to midnight of the same day (just like TRUNC). If the date falls after noon, rounding `SYSDATE` will set it as equal to midnight of the next day.
- If the `SELECT` is issued at 10:00 AM, the values for T2 and T3 will both be equal to midnight of the same day on which the `SELECT` was issued. If the `SELECT` is issued any time after noon, the value of the truncated `SYSDATE` will be midnight of the current day, but the value of the rounded `SYSDATE` will be midnight of the next day.

A number can be added to a column or a variable defined as a date. This operation will add the number of days represented by that number to the given date, producing another date as the result. The number represents the number of days being added, not the number of months.

If a date field were inserted without a time component, the time component would default to zero. For example, if you inserted `04-Jul-07`, then the stored value will be July 4 2007 AD at 0 hours, 0 minutes, and 0 seconds past midnight. Consequently, if you added the number 1.5 to the date, the internal value for the result will be July 5 2007 AD at noon.

Item: 2 (Ref:1Z0-061.5.3.6)

The `transaction` table contains these columns:

```
SQL> DESCRIBE transaction
TRANSACTION_ID NUMBER(9)
TRANS_CODE VARCHAR2(5)
CUST_ACCOUNT VARCHAR2(12)
```

A new standard was adopted in your department affecting reports produced by querying the transaction table. When creating reports, a dash (-) followed by the three characters `ANI` must be appended to all transaction codes that contain only three characters. Any leading 'W' in a transaction code must be removed from the resulting data display.

Which query will return the desired results?

- `SELECT TRIM('W' (RPAD(trans_code, 7, '-ANI')))FROM transaction WHERE LENGTH (trans_code) = 3;`
- `SELECT TRIM('W' FROM (RPAD(trans_code, 3, '-ANI')))FROM transaction WHERE LENGTH (trans_code) = 3;`
- `SELECT TRIM(LEADING 'W' FROM (RPAD(trans_code, 7, '-ANI')))FROM transaction WHERE LENGTH(trans_code) = 3;`
- `SELECT TRIM(LEADING 'W' FROM (RPAD(trans_code, 3, '-ANI'))) FROM transaction WHERE LENGTH(trans_code) = 3;`

Answer:

`SELECT TRIM(LEADING 'W' FROM (RPAD(trans_code, 7, '-ANI')))FROM transaction WHERE LENGTH(trans_code) = 3;`

Explanation:

The following query will return the desired results:

```
SQL> SELECT TRIM(LEADING 'W' FROM (RPAD(trans_code, 7, '-ANI')))
FROM transaction WHERE LENGTH(trans_code) = 3;
```

The query first limits the values it will return. The `WHERE` clause limits the rows returned to only rows that have a `trans_code` with a length of 3. The characters '`-ANI`' are appended to the value using the `RPAD` function. If the first character in the `trans_code` is a '`W`', the `TRIM` function will remove it from it from the query results. The `LEADING` keyword can also be used returning the same results, as in:

```
TRIM(LEADING 'W' FROM (RPAD(trans_code, 7, '-ANI')))
```

The `LENGTH` function returns the number of characters in a value (column or character string). The returned value is of a `NUMBER` data type. If the column or expression is null, `NULL` is returned. The `LENGTH` character function may be used in a `SELECT` list or in a `WHERE` clause condition, such as `WHERE LENGTH(trans_code) = 3`, to restrict the results of a query to columns having a specified length. The syntax of the `LENGTH` function is:

```
LENGTH(column|expression)
```

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

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

The `TRIM` character function is used to trim leading, trailing, or both leading and trailing characters from a value. Character literals must be enclosed in single quotes. When the `LEADING`, `TRAILING`, or `BOTH` keywords are omitted, the default behavior is to trim the leading and trailing characters. The syntax of the `TRIM` function is:

```
TRIM(LEADING|TRAILING|BOTH, trim_character FROM column|expression)
```

The statement that does not include the `FROM` keyword will fail. The `FROM` clause is required for all `SELECT` statements.

The statements containing `SELECT TRIM(LEADING 'W' FROM (RPAD(trans_code, 3, '-ANI')))` and `SELECT TRIM('W' FROM (RPAD(trans_code, 3, '-ANI')))` execute successfully, but do not return the desired results.

Item: 3 (Ref:1Z0-061.5.3.5)

The `Product` table contains these columns:

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

Evaluate these two statements.

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

Statement 2:
```
SQL> SELECT NVL(TO_CHAR(100/quantity), 'none')
FROM Product;
```

Which of the following statements is TRUE?

- ◯ Both statements execute successfully.
- ◯ Statement 1 will 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 will 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 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')
```

Statement 2 does not cause an error when quantity values are null. The use of the `NVL` function accompanied by the `TO_CHAR` conversion function ensures that the text `none` is displayed when the value of `100/quantity` is unknown (NULL). Statement 2 executes successfully, displaying the value '`none`' in the place of null values.

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 of 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 data type. When the expression data types differ, Oracle converts `expression2` to the data type of `expression1` before the two expressions are compared.

Item: 4 (Ref:1Z0-061.5.3.7)

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

Assume the user enters the following SELECT statement to retrieve data from the TEACHER table:

```
SQL> SELECT *
FROM teacher
WHERE INSTR(subject_id, '&1') = 4
AND LOWER(subject_id) LIKE 'HST%';
```

When prompted for the WHERE clause value, you enter an underscore (_).

Which result will this statement provide?

- ◯ It will execute, but it will not retrieve any data.
- ◯ It will display information on all teachers whose subject_id begins with HST_.
- ◯ It will return a syntax error because the TO_CHAR function was not used in the WHERE clause.
- ◯ It will display information on all teachers whose subject_id begins with HST_, regardless of the case in which the subject_id is stored.

Answer:

It will execute, but it will not retrieve any data.

Explanation:

This statement will execute, but it will not retrieve any data. In the second condition of the WHERE clause, the values in the subject_id column are converted to lowercase and then compared to a character string in uppercase, which results in no rows being returned. Since this condition is ALWAYS FALSE, and this condition is joined by an AND with the first condition, the compound condition will be ALWAYS FALSE as well. Consequently, the truth value of the first condition does not need to be evaluated to answer this question. A SELECT statement in which the WHERE clause is ALWAYS FALSE will always give the results "no rows returned".

To display information on only teachers whose subject_id begins with 'HST', you could add AND subject_id LIKE 'HST%' to the WHERE clause. To display the same information regardless of the case in which the subject_id is stored, you could add AND UPPER(subject_id) LIKE 'HST%' to the WHERE clause.

The UPPER character function converts a mixed case or lowercase value (a column or expression) to uppercase. The syntax of the UPPER character function is:

```
UPPER(column|expression)
```

The `LOWER` character function converts a mixed case or uppercase value (a column or expression) to lowercase. The syntax of the `LOWER` character function is:

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

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

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

Item: 5 (Ref:1Z0-061.5.3.2)

The `employee` table contains a column called `first_name` and a column called `last_name`. Both columns have been defined as `VARCHAR2(25)`. You want to create a username for the employee that is comprised of all lowercase characters. It will be derived by taking the first character of the first name followed by the full last name. As an example, if `first_name` is `jOhn` and the `last_name` is `sMitH`, the username would be `jsmith`.

However, there is an 8-character maximum for usernames. Which one of the following `SELECT` statements will correctly produce the username for each employee? (Choose all that apply.)

- ☐ `SELECT first_name, last_name, LOWER(SUBSTR(first_name,1,1))||LOWER(SUBSTR (last_name,1,7)) username FROM employee e1`
- ☐ `SELECT first_name, last_name,LOWER(CONCAT( LTRIM(first_name,1),SUBSTR (last_name,1,7) username FROM employee e2`
- ☐ `SELECT first_name, last_name, LOWER(CONCAT(SUBSTR(first_name,1,1),SUBSTR (last_name,1,7))) username FROM employee e3`
- ☐ `SELECT first_name, last_name,CONCAT(LOWER(SUBSTR(first_name,1,1)),LOWER(SUBSTR (last_name,1,7)) ) username FROM employee e4`
- ☐ `SELECT first_name, last_name, LOWER(CONCAT(SUBSTR(first_name,1,1)||SUBSTR (last_name,1,7))) username FROM employee e5`

Answer:

`SELECT first_name, last_name, LOWER(SUBSTR(first_name,1,1))||LOWER(SUBSTR (last_name,1,7)) username FROM employee e1`

`SELECT first_name, last_name, LOWER(CONCAT(SUBSTR(first_name,1,1),SUBSTR (last_name,1,7))) username FROM employee e3`

`SELECT first_name, last_name,CONCAT(LOWER(SUBSTR(first_name,1,1)),LOWER (SUBSTR(last_name,1,7)) ) username FROM employee e4`

Explanation:

A variety of functions are used in the various answers.

The `LOWER(SUBSTR(first_name,1,1)` expression would extract the first character of the `first_name`. Then that character would be converted to lowercase (if it wasn't already lower case).

The `LOWER(SUBSTR(last_name,1,7)` expression would extract the first seven characters of `last_name`. If there are fewer than seven characters, it would take all that are available. It then converts any uppercase characters to lowercase.

The `LTRIM('abcdefg','ab')` expression will take the first argument (string) and remove zero (0) or more occurrences of the second argument (string), starting at the far left of argument 1. Whatever is left of argument 1 is the value this function returns.

`a || b || c` will concatenate together the strings a, b, and c. This is not the use of a function, but rather the use of the || operator. There is no practical limit to the number of items that can be concatenated together using the ||.

`CONCAT(a,b)` will concatenate together a followed by b. The `CONCAT` function limits you to concatenating two items together at a time.

In the statement `SELECT first_name, last_name,LOWER(CONCAT( LTRIM(first_name,1),SUBSTR (last_name,1,7) username FROM employee e2`, the `LTRIM` function is being used, but it will not return the

first character of `first_name`. Hence, this statement will not produce a username in the required format.

In the statement `SELECT first_name, last_name, LOWER(CONCAT(SUBSTR(first_name,1,1) ||SUBSTR(last_name,1,7))) username FROM employee e5`, the `SUBSTR` function is used properly to extract the appropriate character from `first_name` and the appropriate characters from `last_name`. Then they are properly combined by the `||` operator, resulting in a single value. However, that single value now exists as the argument in a `CONCAT` function. Because the `CONCAT` function requires two arguments, not one, the command contains a syntax error.

All three of the remaining answers are correct because they perform the three tasks required to create the username. These are:

1. Strip of the first character of first name and strip off the first seven characters of last name.
2. Concatenate those two things together.
3. Force the result into lower case.

You can perform step 2 using either the `CONCAT` function or the `||` operator. Step 3 can be performed before step 1, after step 1 but before step 2, or after step 2. You actually should be able to come up with six different ways to write this command to obtain the correct result.

Item: 6 (Ref:1Z0-061.5.2.2)

The current date is January 1, 2009. You need to store this date value:

```
19-OCT-99
```

Which statement about the date format for this value is TRUE?

- ◯ Both the YY and RR date formats will interpret the year as 1999.
- ◯ Both the YY and RR date formats will interpret the year as 2099.
- ◯ The RR date format will interpret the year as 2099, and the YY date format will interpret the year as 1999.
- ◯ The RR date format will interpret the year as 1999, and the YY date format will interpret the year as 2099.

Answer:

The RR date format will interpret the year as 1999, and the YY date format will interpret the year as 2099.

Explanation:

If the current year is 2009, the RR date format will interpret 19-OCT-99 as the year 1999. The century recognized by the RR date format varies depending on the specified two-digit year and the last two digits of the current year. The default date display is `DD-MON-RR`. The RR format allows you to store 21st century dates in the 20th century by specifying only the last two digits of the year, and allows you to store 20th century dates in the 21st century the same way.

The YY date format will interpret `19-OCT-99` as the year 2099. This date format will interpret only the current century, in this example the 21st century.

Item: 7 (Ref:1Z0-061.5.2.5)

The `student` table contains these columns:

```
SQL> DESCRIBE student
ID NUMBER(9)
LAST_NAME VARCHAR2(25)
FIRST_NAME VARCHAR2(25)
ENROLL_DATE DATE
```

You need to display the `enroll_date` values in this format:

```
25th of February 2011
```

Which `SELECT` statement should you use?

- ◯ `SELECT enroll_date('DDspth "of" Month YYYY') FROM student;`
- ◯ `SELECT TO_CHAR(enroll_date, 'ddth "of" Month YYYY') FROM student;`
- ◯ `SELECT TO_CHAR(enroll_date, 'DDTH "of" Month YYYY') FROM student;`
- ◯ `SELECT TO_CHAR(enroll_date, 'DDspth 'of' Month YYYY') FROM student;`

Answer:

```
SELECT TO_CHAR(enroll_date, 'ddth "of" Month YYYY')
FROM student;
```

Explanation:

You should use the following `SELECT` statement:

```
SQL> SELECT TO_CHAR(enroll_date, 'ddth "of" Month YYYY')
FROM student;
```

The `TO_CHAR` function converts the date value to a character value in order to display the date value in a specified format. The `dd` portion of the format model represents the day of the month, followed by `th` (displayed as 25th). `Month` represents the month of the year (February) and `YYYY` represents all of the digits of the year (2011). The use of upper and lower case characters are significant in the format mask.

The `TO_CHAR` conversion function converts a date or number value to a `VARCHAR2` character string using a format model. When a format is not provided, the default date format is used. The syntax of the `TO_CHAR` function is:

```
TO_CHAR(date|number [, 'fmt'])
```

The `SELECT` statement that does not include the `TO_CHAR` function fails because the `TO_CHAR` function is omitted. Without the conversion of the `enroll_date` to a character value, no format can be applied to the display of the query results.

The statement containing `SELECT TO_CHAR(enroll_date, 'DDTH "of" Month YYYY')` returns the enroll date is this format: `25TH of February 2011`.

The statement containing `SELECT TO_CHAR(enroll_date, 'DDspth 'of' Month YYYY')` displays the enroll date is this format: `TWENTY-FIFTH of February 2011`.

Item: 8 (Ref:1Z0-061.5.3.3)

Calculate the value returned by this `SELECT` statement:

```
SQL> SELECT ROUND(16.9) - TRUNC(4.8) - MOD (41,14) FROM dual;
```

- ◯ 2
- ◯ -2
- ◯ 0
- ◯ 13
- ◯ -1

Answer:

0

Explanation:

The correct answer is zero (0).

`ROUND(16.9)` means to round the number to 0 decimal places, or the nearest whole number. That would be 17. This function can have two arguments, but if it only has one then the second argument defaults to zero (0).

`TRUNC(4.8)` means to truncate the number 4.8 to 0 decimal places, or the greatest whole number that is less than or equal to 4.8. That would be 4.

`MOD(41,14)` means to take the remainder when 41 is divided by 14. The number 41 divided by 14 gives an answer of 2 with a remainder of 13, so 13 is the result of this function. The whole number portion of the result of the division problem is discarded, only the remainder is important.

Putting the individual values into the expression yields 17 - 4 - 13, which equals 0.

Since 0 is the right answer, all other values must be incorrect.

Item: 9 (Ref:1Z0-061.5.2.1)

The student table contains these columns:

```
SQL> DESCRIBE student
ID NUMBER(9)
LAST_NAME VARCHAR2(25)
FIRST_NAME VARCHAR2(25)
ENROLL_DATE DATE
```

You need to create a report to display a student's enrollment date and projected graduation date.

These are the desired results:

- Prompt the user for a student ID.
- Display the student's first name, last name, and date of enrollment.
- Display the student's projected graduation date by adding four years to the enrollment date value.

Which statement produces all three of the desired results?

○
```
SELECT CONCAT(INITCAP(first_name), INITCAP(last_name)),
TO_CHAR(ADD_MONTHS(enroll_date, 48), 'DD MONTH YYYY') grad_date
FROM student
WHERE id = &id;
```

○
```
SELECT CONCAT(INITCAP(first_name), INITCAP(last_name)),
TO_CHAR(enroll_date, 'DD MONTH YYYY'), ADD_MONTHS(enroll_date, 48) grad_date
FROM student
WHERE id = &id;
```

○
```
SELECT INITCAP(first_name)||INITCAP(last_name), TO_CHAR(enroll_date, 'DD MONTH
YYYY'),
ADD_YEARS(enroll_date, 4) grad_date
FROM student
WHERE id = &id;
```

○
```
SELECT INITCAP(first_name)||INITCAP(last_name) student_name,
TO_CHAR(enroll_date, 'DD MONTH YYYY') date_enrolled,
TO_CHAR(ADD_MONTHS(enroll_date, 4), 'DD MONTH YYYY') grad_date
FROM student
WHERE id = &id;
```

Answer:

```
SELECT CONCAT(INITCAP(first_name), INITCAP(last_name)),
TO_CHAR(enroll_date, 'DD MONTH YYYY'), ADD_MONTHS(enroll_date, 48) grad_date
FROM student
WHERE id = &id;
```

Explanation:

The following statement produces all three desired results:

```
SQL> SELECT CONCAT(INITCAP(first_name), INITCAP(last_name)),
TO_CHAR(enroll_date, 'DD MONTH YYYY'), ADD_MONTHS(enroll_date, 48) grad_date
FROM student
WHERE id = &id;
```

The user is prompted for the student ID because a substitution variable is used, and the student's first name, last name, and date of enrollment are displayed. The third result is also achieved by using 48 months, which represents four years, with the `ADD_MONTHS` function. This displays the projected graduation date.

The statement that passes a value of `4` to the `ADD_MONTHS` function is incorrect. This statement achieves the first two desired results, but not the third. It will return a projected graduation date that is four months later than the enrollment date.

The following statement achieves the first two desired results, but not the third because the date of enrollment is not queried:

```
SQL> SELECT CONCAT(INITCAP(first_name), INITCAP(last_name)),
TO_CHAR(ADD_MONTHS(enroll_date, 48), 'DD MONTH YYYY') grad_date;
```

The `SELECT` statement containing the `ADD_YEARS` function fails because `ADD_YEARS` is not a valid SQL function.

Item: 10 (Ref:1Z0-061.5.1.1)

Which three function descriptions are TRUE? (Choose three.)

- ☐ The SYSDATE function returns the local machine date and time.
- ☐ The NVL single-row function can be used on VARCHAR2 columns.
- ☐ The LENGTH character function returns the number of characters in an expression.
- ☐ The ROUND number function returns a number rounded to the specified column value.
- ☐ The TRUNC date function returns a date with the time portion of the day truncated to the specified format unit.
- ☐ The SUBSTR character function replaces a portion of a string, beginning at a defined character position for a defined length.

Answer:

The NVL single-row function can be used on VARCHAR2 columns.

The LENGTH character function returns the number of characters in an expression.

The TRUNC date function returns a date with the time portion of the day truncated to the specified format unit.

Explanation:

The following function descriptions are true:

- The NVL single-row function can be used on VARCHAR2 columns.
- The LENGTH character function returns the number of characters in an expression.
- The TRUNC date function returns a date with the time portion of the day truncated to the specified format unit.

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 data type. When the expression data types differ, Oracle converts expression2 to the data type of expression1 before the two expressions are compared.

The LENGTH function returns the number of characters in an expression (column or character string). The value returned is of NUMBER data type. If the column or expression is null, NULL is returned. The syntax of the LENGTH function is:

```
LENGTH(column|expression)
```

The TRUNC date function returns a date with the time portion of the day truncated to the specified format unit. If no format model ('fmt') is provided, the date is truncated to the nearest day. TRUNC may be used as either a date or a number function. The syntax of the TRUNC function is:

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

The `SYSDATE` function returns the current database server date and time as a `DATE` data type. It does not return the host machine date and time. This function requires no arguments.

The `ROUND` number function does not return a number rounded to the specified column value. The `ROUND` function is used to round values to a specified number of decimal places. 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` function is:

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

The `SUBSTR` character function does not replace a portion of a string but instead returns a portion of a string, beginning at a specified position (m) and ending at a specified substring length (n). A position of zero (0) is treated as a 1. When the 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])
```

Item: 11 (Ref:1Z0-061.5.3.8)

Seniority is based on the number of years a student has been enrolled at the university. You must create a report that displays each student's name, ID number, and the number of years enrolled. The years enrolled must be rounded to a whole number, based on the number of months from the date enrolled until today.

Which statement produces the required results?

- ◯
```
SELECT first_name||', '||last_name "Student Name", id "Id", enroll_date, ROUND
(SYSDATE) - ROUND(enroll_date) "Seniority"
FROM student;
```
- ◯
```
SELECT first_name||', '||last_name "Student Name", id "Id", enroll_date,

(ROUND(SYSDATE) - ROUND(enroll_date)) / 12 "Seniority"
FROM student;
```
- ◯
```
SELECT first_name||', '||last_name "Student Name", id "Id", enroll_date,

TRUNC(SYSDATE, 'YY') - TRUNC(enroll_date, 'YY') "Seniority"
FROM student;
```
- ◯
```
SELECT first_name||', '||last_name "Student Name", id "Id", enroll_date,

ROUND(MONTHS_BETWEEN(SYSDATE, enroll_date) / 12) "Seniority"
FROM student;
```

Answer:

```
SELECT first_name||', '||last_name "Student Name", id "Id", enroll_date,

ROUND(MONTHS_BETWEEN(SYSDATE, enroll_date) / 12) "Seniority"
FROM student;
```

Explanation:

The following statement produces the required results:

```
SQL> SELECT first_name||', '||last_name "Student Name", id "Id", enroll_date,
ROUND(MONTHS_BETWEEN(SYSDATE, enroll_date) / 12) "Seniority"
FROM student;
```

The `MONTHS_BETWEEN` function determines the number of months between two dates. In this scenario, the number of months between today, `SYSDATE`, and the date a student is enrolled, `enroll_date`, will be determined. The resulting value is then divided by 12 to return a year value. The `ROUND` function rounds the year value to the nearest integer.

When you are using the `MONTHS_BETWEEN` date function, the value returned is positive when `date1` is later than `date2`. The value returned is negative when `date1` is earlier than `date2`. The syntax of the `MONTHS_BETWEEN` function is:

```
MONTHS_BETWEEN(date1, date2)
```

The `SYSDATE` function returns the current data and time as a `DATE` data type. This function requires no arguments and returns the date and time of the database server.

The `ROUND` number function is used to round values to a specified decimal value. When an integer representing the number of decimal places to be used (n) is not provided, the number is rounded to 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 `TRUNC` date function returns a date with the time portion of the day truncated to the specified format (`'fmt'`). If no format is provided, the date is truncated to the nearest day. `TRUNC` may be used as either a date or a number function. The syntax of the `TRUNC` date function is:

```
TRUNC(date, [, 'fmt'])
```

The statement that uses `ROUND(SYSDATE) - ROUND(enroll_date)` will round the date values to the nearest day and then subtract today's date from the date enrolled.

The statement that uses `(ROUND(SYSDATE) - ROUND(enroll_date)) / 12` will round the date values to the nearest day and then subtract today's date from the date enrolled. This value will be divided by 12.

The statement that uses `TRUNC(SYSDATE, 'YY') - TRUNC(enroll_date, 'YY')` will truncate each value to a two-digit year and subtract the date enrolled from today's date. This will not retrieve accurate results because only a two-digit year is used.

Item: 12 (Ref:1Z0-061.5.2.4)

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 report that shows each employee's name and the number of years each employee has worked for the company. Assume continuous employment since their hire dates. For this report, the employees should be sorted such that the person with the most seniority is first. Years of employment should be rounded to one decimal place.

A few employees do not yet have a hire date because they just received a job offer and an actual `HIREDATE` has not yet been determined. Consequently, the `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((HIREDATE - SYSDATE)/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
```

Explanation:

The following statement will produce the desired results:

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

The `hiredate` will be subtracted from the `sysdate` giving the number of days since their hire date. Dividing that by 365 will convert the answer from days to years. The `ROUND` function correctly rounds the number of years to 1 significant digit. The `NVL` function guarantees that if no value existed for `hireday`, the result would be displayed as a 0.

If we sorted by `hiredate DESC`, the order of the names would be from the most recently hired employee to the least recently hired, just the opposite of what was required.

If you `ORDER BY 3`, you are ordering by the 3rd column, which contains (after the computations are performed) the years of seniority. The default order is ascending, so again, the output would not be in the correct order.

If you subtract `hiredate - sysdate` the result will be a negative number of days. This will become a negative number of years. The output would not appear as requested.

Item: 13 (Ref:1Z0-061.5.2.3)

The `employee` table contains these columns:

```
SQL> DESCRIBE employee
emp_id NUMBER(9)
LAST_NAME VARCHAR2(20)
FIRST_NAME VARCHAR2(20)
COMM_PCT NUMBER(2)
```

You need to display the commission percentage for each employee followed by a percent sign (`%`). If an employee does not receive a commission, the output should display `No Comm`. Employee commissions do not exceed 20 percent.

Which statement should you use to achieve these results?

- `SELECT emp_id, last_name, NVL(comm_pct||'%', 'No Comm') FROM employee;`
- `SELECT emp_id, last_name, NVL(TO_CHAR(comm_pct||'%'), 'No Comm') FROM employee;`
- `SELECT emp_id, last_name, RPAD(NVL(TO_CHAR(comm_pct), 'No Comm'), '%') FROM employee;`
- None of the statements returns the desired results.

Answer:

None of the statements returns the desired results.

Explanation:

None of the statements returns the desired results. Three functions are needed to return the desired results: the `TO_CHAR`, `NVL`, and `RPAD` functions. The `TO_CHAR` function converts any non-null `comm_pct` values to a `VARCHAR2` character string. The `NVL` function converts any null `comm_pct` values to the character string `No Comm`. The `RPAD` function appends a percent sign to the end of the `comm_pct` value as the third character. To return the desired results, use this `SELECT` statement:

```
SQL> SELECT emp_id, last_name, RPAD(NVL(TO_CHAR(comm_pct,'99'), 'No Comm'), 3,'% ')
FROM employee;
```

The `SELECT` clause that contains `NVL(comm_pct||'%', 'No Comm')` does not return the desired results because the percent sign is appended to the `commission_pct` column in the `SELECT` clause. The value `%` is displayed when there is a null value.

The `SELECT` clause that contains `NVL(TO_CHAR(comm_pct||'%'), 'No Comm')` fails because of the improper use of the concatenation operator within the `TO_CHAR` function parameters.

The `SELECT` clause that contains `RPAD(NVL(TO_CHAR(comm_pct), 'No Comm'), '%')` is incorrect because it does not pass the `RPAD` function the correct number of arguments.

The `TO_CHAR` function is used to convert a number or date value to a `VARCHAR2` character string using a format model ('`fmt`'). The syntax for the `TO_CHAR` function is:

```
TO_CHAR(number, 'fmt')
```

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

```
NVL(column|expression1, expression2)
```

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

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

Item: 14 (Ref:1Z0-061.5.3.4)

The table called `Customer` contains a column called `city`, which is defined as `VARCHAR2(15)`. The application developer believes that a couple of cities actually are stored in a truncated manner because the name of the city, when written out in English, is more than 15 characters. Which `SELECT` statement will find the `customer_name` and `city` where all 15 characters are being used to store the name of the city?

- ◯ `SELECT customer_name, city FROM Customer WHERE city(LENGTH) = 15`
- ◯ `SELECT customer_name, city FROM Customer WHERE LENGTH(city) > 15`
- ◯ `SELECT customer_name, city FROM Customer WHERE LENGTH(city) = 15`
- ◯ `SELECT customer_name, city FROM Customer WHERE city(LENGTH) = 15`
- ◯ It is impossible to answer this question because the column `city` is of variable length due to the `VARCHAR2` column definition.

Answer:

```
SELECT customer_name, city FROM Customer WHERE LENGTH(city) = 15
```

Explanation:

The `SELECT` statement with the conditional clause `WHERE LENGTH(city)= 15` will find the customer name and city name of each row in the table where all 15 characters of the `city` column are being used. The function `LENGTH` allows you to find the number of characters stored in a particular column or variable. The correct syntax of the `LENGTH` function is:

```
LENGTH(argument_name)
```

where `argument_name` is a character string.

If the value of `LENGTH(city)` equals 15, that implies that the maximum number of characters defined in the `VARCHAR2` data type are being used to store the value of city. The value of city could never be equal to anything larger than 15 due to the column data type of `VARCHAR2`(15). However, it could be less than 15, since when you store `ROME` as the value for `city`, it will only use four characters and the `LENGTH` function would return a `4`. `VARCHAR2` data types do not store blanks or any other character to fill out the field to 15 characters. This is the main advantage of this data type.

The conditional clauses which use the format `city(LENGTH)` are incorrect because that would be interpreted as a function called `city` with an argument called `LENGTTH`.

The conditional clause which searches for cities with a length value greater than 15 is incorrect because the number of characters in a column defined as `VARCHAR2`(15) cannot exceed 15.