Free Questions for C_ABAPD_2309 by vceexamstest

Shared by Sharpe on 12-12-2023

For More Free Questions and Preparation Resources

Check the Links on Last Page

Question 1

Question Type: MultipleChoice

Which of the following are ABAP Cloud Development Model rules?

Options:

A- Note: There are 2 correct answers to this question.

B- Use public SAP APIs and SAP extension points.

C- Build ABAP RESTful application programming model-based services.

D- Reverse modifications when a suitable public SAP API becomes available.

E- Build ABAP reports with either ABAP List Viewer (ALV) or SAP Fiori.

Answer:

A

Explanation:

Use public SAP APIs and SAP extension points. This rule ensures that the ABAP Cloud code is stable, reliable, and compatible with the SAP solutions and the cloud operations. Public SAP APIs and SAP extension points are the only allowed interfaces and objects to access the SAP platform and the SAP applications. They are documented, tested, and supported by SAP.They also guarantee the lifecycle stability and the upgradeability of the ABAP Cloud code1.

Build ABAP RESTful application programming model-based services. This rule ensures that the ABAP Cloud code follows the state-of-the-art development paradigm for building cloud-ready business services. The ABAP RESTful application programming model (RAP) is a framework that provides a consistent end-to-end programming model for creating, reading, updating, and deleting (CRUD) business dat

a. RAP also supports draft handling, authorization checks, side effects, validations, and custom actions.RAP exposes the business services as OData services that can be consumed by SAP Fiori apps or other clients2.

Question 2

Question Type: **MultipleChoice**

Given the following Core Data Service View Entity Data Definition:

1 @AccessControl.authorizationCheck: #NOT_REQUIRED

2 DEFINE VIEW ENTITY demo_flight_info_join

3 AS SELECT

4 FROM scarr AS a

5 LEFT OUTER JOIN scounter AS c

6 LEFT OUTER JOIN sairport AS p

7 ON p.id = c.airport

8 ON a.carrid = c.carrid

9 {

10 a.carrid AS carrier_id,

11 p.id AS airport_id,

12 c.countnum AS counter_number

13 }

In what order will the join statements be executed?

Options:

A- scarr will be joined with scounter first and the result will be joined with sairport.

B- sairport will be joined to scounter first and the result will be joined with scarr.

C- scarr will be joined with sairport first and the result will be joined with scounter.

D- scounter will be joined to sairport first and the result will be joined with scarr.

Answer:

A

Explanation:

The order in which the join statements will be executed is:

scarr will be joined with scounter first and the result will be joined with sairport.

This is because the join statements are nested from left to right, meaning that the leftmost data source is joined with the next data source, and the result is joined with the next data source, and so on. The join condition for each pair of data sources is specified by the ON clause that follows the data source name. The join type for each pair of data sources is specified by the join operator that precedes the data source name. In this case, the join operator is LEFT OUTER JOIN, which means that all the rows from the left data source are included in the result, and only the matching rows from the right data source are included. If there is no matching row from the right data source, the corresponding fields are filled with initial values1.

Therefore, the join statements will be executed as follows:

First, scarr AS a will be joined with scounter AS c using the join condition a.carrid = c.carrid. This means that all the rows from scarr will be included in the result, and only the rows from scounter that have the same value for the carrid field will be included. If there is no matching row from scounter, the countnum field will be filled with an initial value.

Second, the result of the first join will be joined with sairport AS p using the join condition p.id = c.airport. This means that all the rows from the first join will be included in the result, and only the rows from sairport that have the same value for the id field as the airport field from the first join will be included. If there is no matching row from sairport, the id field will be filled with an initial value.

Question 3

Question Type: MultipleChoice

You have two internal tables itab1 and itab2.What is true for using the expression itab1 = corresponding #(itab2)? Note: There are 2 correct answers to this question.

Options:

A- Fields with the same name but with different types may be copied from itab2 to itab1.

B- itab1 and itab2 must have at least one field name in common.

C- Fields with the same name and the same type will be copied from itab2 to itab1.

D- itab1 and itab2 must have the same data type.

Answer:

B, C

Explanation:

The expression itab1 = corresponding #(itab2) is a constructor expression with the component operator CORRESPONDING that assigns the contents of the internal table itab2 to the internal table itab1. The following statements are true for using this expression:

B: itab1 and itab2 must have at least one field name in common. This is because the component operator CORRESPONDING assigns the identically named columns of itab2 to the identically named columns of itab1 by default, according to the rules of MOVE-CORRESPONDING for internal tables.If itab1 and itab2 do not have any field name in common, the expression will not assign any value to itab1 and it will remain initial or unchanged1

C: Fields with the same name and the same type will be copied from itab2 to itab1. This is because the component operator CORRESPONDING assigns the identically named columns of itab2 to the identically named columns of itab1 by default, according to the rules of MOVE-CORRESPONDING for internal tables.If the columns have the same name but different types, the assignment will try to perform a conversion between the types, which may result in a loss of precision, a truncation, or a runtime error, depending on the types involved1

The following statements are false for using this expression:

A: Fields with the same name but with different types may be copied from itab2 to itab1. This is not true, as explained in statement C.The assignment will try to perform a conversion between the types, which may result in a loss of precision, a truncation, or a runtime error, depending on the types involved1

D: itab1 and itab2 must have the same data type. This is not true, as the component operator CORRESPONDING can assign the contents of an internal table of one type to another internal table of a different type, as long as they have at least one field name in common. The target type of the expression is determined by the left-hand side of the assignment, which is itab1 in this case.The

expression will create an internal table of the same type as itab1 and assign it to itab11

Question 4

Question Type: MultipleChoice

Refer to exhibit.

when you attempt to activate the definition, what will be the response?

Options:

A- Activation error because the field names of the union do not match

B- Activation error because the field types of the union do not match

C- Activation error because the key fields of the union do not match

D- Activation successful

Answer:

A

Explanation:

The response will be an activation error because the field names of the union do not match. This is because the field names of the union must match in order for the definition to be activated. The union operator combines the result sets of two or more queries into a single result set.The queries that are joined by the union operator must have the same number and type of fields, and the fields must have the same names1. In the given code, the field names of the union do not match, because the first query has the fields carrname, connid, cityfrom, and cityto, while the second query has the fields carrname, carrier_id, cityfrom, and cityto. The field connid in the first query does not match the field carrier_id in the second query. Therefore, the definition cannot be activated.

Question 5

Question Type: MultipleChoice

After you created a database table in the RESTful Application Programming model, what do you create next?

Options:

A- A metadata extension

B- A projection view

C- A data model view

D- A service definition

Answer:

B

Explanation:

After you created a database table in the RESTful Application Programming model (RAP), the next step is to create a projection view on the database table. A projection view is a CDS artefact that defines a view on one or more data sources, such as tables, views, or associations. A projection view can select, rename, or aggregate the fields of the data sources, but it cannot change the properties of the fields, such as whether they are read-only or not. The properties of the fields are inherited from the data sources or the behaviour definitions of the business objects12. For example:

The following code snippet defines a projection view ZI_AGENCY on the database table /DMO/AGENCY:

define view ZI_AGENCY as select from /dmo/agency { key agency_id, agency_name, street, city, region, postal_code, country, phone_number, url }

The projection view is used to expose the data of the database table to the service definition, which is the next step in the RAP. The service definition is a CDS artefact that defines the interface and the binding of a service. A service is a CDS entity that exposes the data and the functionality of one or more business objects as OData, InA, or SQL services. A service definition can specify the properties of the fields of a service, such as whether they are filterable, sortable, or aggregatable12. For example:

The following code snippet defines a service definition ZI_AGENCY_SRV that exposes the projection view ZI_AGENCY as an OData service:

define service ZI_AGENCY_SRV { expose ZI_AGENCY as Agency; }

You cannot do any of the following:

A) A metadata extension: A metadata extension is a CDS artefact that defines additional annotations for a CDS entity, such as a business object, a service, or a projection view. A metadata extension can specify the properties of the fields of a CDS entity for UI or analytical purposes, such as whether they are visible, editable, or hidden. However, a metadata extension is not the next step after creating a database table in the RAP, as it is not required to expose the data of the database table to the service definition. A metadata extension can be created later to customize the UI or analytical application that uses the service12.

C) A data model view: A data model view is a CDS artefact that defines a view on one or more data sources, such as tables, views, or associations. A data model view can select, rename, or aggregate the fields of the data sources, and it can also change the properties of the fields, such as whether they are read-only or not. The properties of the fields are defined by the annotations or the behaviour definitions of the data model view. A data model view is used to define the data model of a business object, which is a CDS entity that represents a business entity or concept, such as a customer, an order, or a product. However, a data model view is not the next step after creating a database table in the RAP, as it is not required to expose the data of the database table to the service definition. A data

model view can be created later to define a business object that uses the database table as a data source12.

D) A service definition: A service definition is a CDS artefact that defines the interface and the binding of a service. A service is a CDS entity that exposes the data and the functionality of one or more business objects as OData, InA, or SQL services. A service definition can specify the properties of the fields of a service, such as whether they are filterable, sortable, or aggregatable. However, a service definition is not the next step after creating a database table in the RAP, as it requires a projection view or a data model view to expose the data of the database table. A service definition can be created after creating a projection view or a data model view on the database table12.

Question 6

Question Type: **MultipleChoice**

Which of the following string functions are predicate functions? Note: There are 2 correct answers to this question.

Options:

A- find_any_not_of()

B- contains_any_of()

C- count_any_of()

D- matchesQ

Answer:

B, D

Explanation:

String functions are expressions that can be used to manipulate character-like data in ABAP. String functions can be either predicate functions or non-predicate functions. Predicate functions are string functions that return a truth value (true or false) for a condition of the argument text.Non-predicate functions are string functions that return a character-like result for an operation on the argument text1.

The following string functions are predicate functions:

B) contains_any_of(): This function returns true if the argument text contains at least one of the characters specified in the character set. For example, the following expression returns true, because the text 'ABAP' contains at least one of the characters 'A', 'B', or 'C':

contains_any_of(val = 'ABAP' set = 'ABC').

D) matches(): This function returns true if the argument text matches the pattern specified in the regular expression. For example, the following expression returns true, because the text 'ABAP' matches the pattern that consists of four uppercase letters:

matches(val = 'ABAP' regex = '[A-Z]{4}').

The following string functions are not predicate functions, because they return a character-like result, not a truth value:

A) find_any_not_of(): This function returns the position of the first character in the argument text that is not contained in the character set. If no such character is found, the function returns 0. For example, the following expression returns 3, because the third character of the text 'ABAP' is not contained in the character set 'ABC':

find_any_not_of(val = 'ABAP' set = 'ABC').

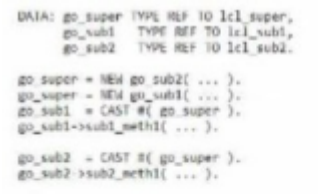
C) count_any_of(): This function returns the number of characters in the argument text that are contained in the character set. For example, the following expression returns 2, because there are two characters in the text 'ABAP' that are contained in the character set 'ABC':

count_any_of(val = 'ABAP' set = 'ABC').

Question 7

Question Type: **MultipleChoice**

Exhibit:



```
DATA: go_super TYPE REF TO lcl_super,
      go_sub1  TYPE REF TO lcl_sub1,
      go_sub2  TYPE REF TO lcl_sub2.

go_super = NEW go_sub2( ... ).
go_super = NEW go_sub1( ... ).
go_sub1  = CAST #( go_super ).
go_sub1->sub1_meth1( ... ).

go_sub2  = CAST #( go_super ).
go_sub2->sub2_meth1( ... ).
```

With Icl_super being superclass for Icl_subl and Icl_sub2 and with methods subl_methl and sub2_methl being subclass-specific methods of Id_subl or Icl_sub2, respectivel. What will happen when executing these casts? Note:

There are 2 correct answers to this question

Options:

A- go subl = CAST # go super), will not work

B- go_sub2 = CAST # go super), will work. go_subl CAST #go_super), will work

C- go_sub2 = CAST #(go_super). will not work.] go sub2->sub2 meth 1(...). will work

D- go_subl->subl_meth !(...)* w'll work.

Answer:

A, D

Explanation:

The following are the explanations for each statement:

A: This statement is correct. go_subl = CAST #(go_super) will not work. This is because go_subl is a data object of type REF TO cl_subl, which is a reference to the subclass cl_subl. go_super is a data object of type REF TO cl_super, which is a reference to the superclass cl_super. The CAST operator is used to perform a downcast or an upcast of a reference variable to another reference variable of a compatible type. A downcast is a conversion from a more general type to a more specific type, while an upcast is a conversion from a more specific type to a more general type. In this case, the CAST operator is trying to perform a downcast from go_super to go_subl, but this is not possible, as go_super is not pointing to an instance of cl_subl, but to an instance of cl_super.Therefore, the CAST operator will raise an exception CX_SY_MOVE_CAST_ERROR at runtime12

B: This statement is incorrect. go_sub2 = CAST #(go_super) will work. go_subl = CAST #(go_super) will not work. This is because go_sub2 is a data object of type REF TO cl_sub2, which is a reference to the subclass cl_sub2. go_super is a data object of type REF TO cl_super, which is a reference to the superclass cl_super. The CAST operator is used to perform a downcast or an upcast of a reference variable to another reference variable of a compatible type. A downcast is a conversion from a more general type to a more specific type, while an upcast is a conversion from a more specific type to a more general type. In this case, the CAST operator is trying to perform a downcast from go_super to go_sub2, and this is possible, as go_super is pointing to an instance of cl_sub2, which is a subclass of cl_super. Therefore, the CAST operator will assign the reference of go_super to go_sub2 without raising an exception.However, the CAST operator will not work for go_subl, as explained in statement A12

C: This statement is incorrect. go_sub2 = CAST #(go_super) will work. go_sub2->sub2_meth1(...) will not work. This is because go_sub2 is a data object of type REF TO cl_sub2, which is a reference to the subclass cl_sub2. go_super is a data object of type REF TO cl_super, which is a reference to the superclass cl_super. The CAST operator is used to perform a downcast or an upcast of a reference variable to another reference variable of a compatible type. A downcast is a conversion from a more general type to a more specific type, while an upcast is a conversion from a more specific type to a more general type. In this case, the CAST operator is trying to perform a downcast from go_super to go_sub2, and this is possible, as go_super is pointing to an instance of cl_sub2, which is a subclass of cl_super. Therefore, the CAST operator will assign the reference of go_super to go_sub2 without raising an exception. However, the method call go_sub2->sub2_meth1(...) will not work, as sub2_meth1 is a subclass-specific method of cl_sub2, which is not inherited by cl_super.Therefore, the method call will raise an exception CX_SY_DYN_CALL_ILLEGAL_METHOD at runtime123

D: This statement is correct. go_subl->subl_meth1(...) will work. This is because go_subl is a data object of type REF TO cl_subl, which is a reference to the subclass cl_subl. subl_meth1 is a subclass-specific method of cl_subl, which is not inherited by cl_super.Therefore, the method call go_subl->subl_meth1(...) will work, as go_subl is pointing to an instance of cl_subl, which has the method subl_meth1123

Question 8

Question Type: **MultipleChoice**

In a subclass subl you want to redefine a component of a superclass superl. How do you achieve this? Note: There are 2 correct answers to this question.

Options:

A- You add the clause REDEFINITION to the component in subl.

B- You implement the redefined component for a second time in superl.

C- You implement the redefined component in subl.

D- You add the clause REDEFINITION to the component in superl.

Answer:

A, C

Explanation:

To redefine a component of a superclass in a subclass, you need to do the following12:

You add the clause REDEFINITION to the component declaration in the subclass. This indicates that the component is inherited from the superclass and needs to be reimplemented in the subclass. The redefinition must happen in the same visibility section as the component declaration in the superclass. For example, if the superclass has a public method m1, the subclass must also declare the redefined method m1 as public with the REDEFINITION clause.

You implement the redefined component in the subclass. This means that you provide the new logic or behavior for the component that is specific to the subclass. The redefined component in the subclass will override the original component in the superclass when the subclass object is used. For example, if the superclass has a method m1 that returns 'Hello', the subclass can redefine the method m1 to return 'Hi' instead.

You cannot do any of the following:

You implement the redefined component for a second time in the superclass. This is not possible, because the superclass already has an implementation for the component that is inherited by the subclass. The subclass is responsible for providing the new implementation for the redefined component, not the superclass.

You add the clause REDEFINITION to the component in the superclass. This is not necessary, because the superclass does not need to indicate that the component can be redefined by the subclass. The subclass is the one that needs to indicate that the component is redefined by adding the REDEFINITION clause to the component declaration in the subclass.

Question 9

Question Type: **MultipleChoice**

In RESTful Application Programming, a business object contains which parts? Note: There are 2 correct answers to this question.

Options:

A- CDS view

B- Behavior definition

C- Authentication rules

D- Process definition

Answer:

A, B

Explanation:

In RESTful Application Programming, a business object contains two main parts: a CDS view and a behavior definition1.

A) CDS view: A CDS view is a data definition that defines the structure and the data source of a business object. A CDS view can consist of one or more entities that are linked by associations or compositions. An entity is a CDS view element that represents a node or a projection of a business object.An entity can have various annotations that define the metadata and the semantics of the business object2.

B) Behavior definition: A behavior definition is a source code artifact that defines the behavior and the validation rules of a business object. A behavior definition can specify the standard CRUD (create, read, update, delete) operations, the draft handling, the authorization checks, and the side effects for a business object.A behavior definition can also define custom actions, validations, and determinations that implement the business logic of a business object3.

The following are not parts of a business object in RESTful Application Programming, because:

C) Authentication rules: Authentication rules are not part of a business object, but part of a service binding. A service binding is a configuration artifact that defines how a business object is exposed as an OData service.A service binding can specify the authentication

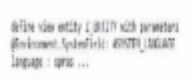
method, the authorization scope, the protocol version, and the service options for the OData service4.

D) Process definition: Process definition is not part of a business object, but part of a workflow. A workflow is a business process that orchestrates the tasks and the events of a business object. A workflow can be defined using the Workflow Editor in the SAP Business Application Studio or the SAP Web IDE.A workflow can use the business object's APIs to trigger or consume events, execute actions, or read or update data5.

Question 10

Question Type: MultipleChoice

What are the effects of this annotation? Note: There are 2 correct answers to this question.



Options:

A- The value of sy-langu will be passed to the CDS view automatically both when you use the -1 CDS view in ABAP and in another CDS view entity (view on view).

B- You can still override the default value with a value of your own.

C- The value of sy-langu will be passed to the CDS view automatically when you use the CDS view in ABAP but not when you use it in another view entity

D- It is no longer possible to pass your own value to the parameter.

Answer:

A, B

Explanation:

The annotation @Environment.systemField: #LANGUAGE is used to assign the ABAP system field sy-langu to an input parameter of a CDS view or a CDS table function. This enables the implicit parameter passing in Open SQL, which means that the value of sy-langu will be automatically passed to the CDS view without explicitly specifying it in the WHERE clause.This also applies to the CDS views that use the annotated CDS view as a data source, which means that the value of sy-langu will be propagated to the nested CDS views (view on view)12. For example:

The following code snippet defines a CDS view ZI_FLIGHT_TEXTS with an input parameter p_langu that is annotated with @Environment.systemField: #LANGUAGE:

define view ZI_FLIGHT_TEXTS with parameters p_langu : syst_langu @<Environment.systemField: #LANGUAGE as select from sflight left outer join scarr on sflight.carrid = scarr.carrid left outer join stext on scarr.carrid = stext.carrid { sflight.carrid, sflight.connid, sflight.fldate, scarr.carrname, stext.text as carrtext } where stext.langu = :p_langu

The following code snippet shows how to use the CDS view ZI_FLIGHT_TEXTS in ABAP without specifying the value of p_langu in the WHERE clause. The value of sy-langu will be automatically passed to the CDS view:

SELECT carrid, connid, fldate, carrname, carrtext FROM zi_flight_texts INTO TABLE @DATA(lt_flights).

The following code snippet shows how to use the CDS view ZI_FLIGHT_TEXTS in another CDS view ZI_FLIGHT_REPORT. The value of sy-langu will be automatically passed to the nested CDS view ZI_FLIGHT_TEXTS:

define view ZI_FLIGHT_REPORT with parameters p_langu : syst_langu @<Environment.systemField: #LANGUAGE as select from zi_flight_texts(p_langu) { carrid, connid, fldate, carrname, carrtext, count(*) as flight_count } group by carrid, connid, fldate, carrname, carrtext

The annotation @Environment.systemField: #LANGUAGE does not prevent the possibility of overriding the default value with a value of your own. You can still specify a different value for the input parameter p_langu in the WHERE clause, either in ABAP or in another CDS view.This will override the value of sy-langu and pass the specified value to the CDS view12. For example:

The following code snippet shows how to use the CDS view ZI_FLIGHT_TEXTS in ABAP with a specified value of p_langu in the WHERE clause. The value 'E' will be passed to the CDS view instead of the value of sy-langu:

SELECT carrid, connid, fldate, carrname, carrtext FROM zi_flight_texts WHERE p_langu = 'E' INTO TABLE @DATA(lt_flights).

The following code snippet shows how to use the CDS view ZI_FLIGHT_TEXTS in another CDS view ZI_FLIGHT_REPORT with a specified value of p_langu in the WHERE clause. The value 'E' will be passed to the nested CDS view ZI_FLIGHT_TEXTS instead of the

value of sy-langu:

define view ZI_FLIGHT_REPORT with parameters p_langu : syst_langu @<Environment.systemField: #LANGUAGE as select from zi_flight_texts(p_langu) { carrid, connid, fldate, carrname, carrtext, count(*) as flight_count } where p_langu = 'E' group by carrid, connid, fldate, carrname, carrtext

Question 11

Question Type: MultipleChoice

What are some characteristics of secondary keys for internal tables? Note: There are 3 correct answers to this question.

Options:

A- Secondary keys must be chosen explicitly when you actually read from an internal table.

B- Multiple secondary keys are allowed for any kind of internal table.

C- Hashed secondary keys do NOT have to be unique.

D- Sorted secondary keys do NOT have to be unique.

E- Secondary keys can only be created for standard tables.

Answer:

A, B, D

Explanation:

Secondary keys are additional keys that can be defined for internal tables to optimize the access to the table using fields that are not part of the primary key. Secondary keys can be either sorted or hashed, depending on the table type and the uniqueness of the key.Secondary keys have the following characteristics1:

A) Secondary keys must be chosen explicitly when you actually read from an internal table. This means that when you use a READ TABLE or a LOOP AT statement to access an internal table, you have to specify the secondary key that you want to use with the USING KEY addition. For example, the following statement reads an internal table itab using a secondary key sec_key:

READ TABLE itab USING KEY sec_key INTO DATA(wa).

If you do not specify the secondary key, the system will use the primary key by default2.

B) Multiple secondary keys are allowed for any kind of internal table. This means that you can define more than one secondary key for an internal table, regardless of the table type. For example, the following statement defines an internal table itab with two secondary keys sec_key_1 and sec_key_2:

DATA itab TYPE SORTED TABLE OF ty_itab WITH NON-UNIQUE KEY sec_key_1 COMPONENTS field1 field2 sec_key_2 COMPONENTS field3 field4.

You can then choose which secondary key to use when you access the internal table1.

D) Sorted secondary keys do NOT have to be unique. This means that you can define a sorted secondary key for an internal table that allows duplicate values for the key fields. A sorted secondary key maintains a predefined sorting order for the internal table, which is defined by the key fields in the order in which they are specified. For example, the following statement defines a sorted secondary key sec_key for an internal table itab that sorts the table by field1 in ascending order and field2 in descending order:

DATA itab TYPE STANDARD TABLE OF ty_itab WITH NON-UNIQUE SORTED KEY sec_key COMPONENTS field1 ASCENDING field2 DESCENDING.

You can then access the internal table using the sorted secondary key with a binary search algorithm, which is faster than a linear search3.

The following are not characteristics of secondary keys for internal tables, because:

C) Hashed secondary keys do NOT have to be unique. This is false because hashed secondary keys must be unique. This means that you can only define a hashed secondary key for an internal table that does not allow duplicate values for the key fields. A hashed secondary key does not have a predefined sorting order for the internal table, but uses a hash algorithm to store and access the table rows. For example, the following statement defines a hashed secondary key sec_key for an internal table itab that hashes the table by field1 and field2:

DATA itab TYPE STANDARD TABLE OF ty_itab WITH UNIQUE HASHED KEY sec_key COMPONENTS field1 field2.

You can then access the internal table using the hashed secondary key with a direct access algorithm, which is very fast.

E) Secondary keys can only be created for standard tables. This is false because secondary keys can be created for any kind of internal table, such as standard tables, sorted tables, and hashed tables. However, the type of the secondary key depends on the type of the internal table.For example, a standard table can have sorted or hashed secondary keys, a sorted table can have sorted secondary keys, and a hashed table can have hashed secondary keys1.

Question 12

Question Type: MultipleChoice

What would be the correct expression to change a given string value 'mr joe doe' into 'JOE' in an ABAP SQL field list?

Options:

A- SELECT FROM TABLE dbtabl FIELDS
Of1,
upper(left('mr joe doe', 6)) AS f2_up_left, f3,

B- SELECT FROM TABLE dbtabl FIELDS
Of1,
left(lower(substring('mr joe doe', 4, 3)), 3) AS f2_left_lo_sub, f3,

C- SELECT FROM TABLE dbtabl FIELDS
Of1,
substring(upper('mr joe doe'), 4, 3) AS f2_sub_up, f3,...

D- SELECT FROM TABLE dbtabl FIELDS
Of1,
substring(lower(upper('mr joe doe')), 4, 3) AS f2_sub_lo_up, f3,

Answer:

C

Explanation:

The correct expression to change a given string value 'mr joe doe' into 'JOE' in an ABAP SQL field list is C.SELECT FROM TABLE dbtabl FIELDS Of1, substring(upper('mr joe doe'), 4, 3) AS f2_sub_up, f3,... This expression uses the following SQL functions for strings12:

upper: This function converts all lowercase characters in a string to uppercase. For example, upper('mr joe doe') returns 'MR JOE DOE'.

substring: This function returns a substring of a given string starting from a specified position and with a specified length. For example, substring('MR JOE DOE', 4, 3) returns 'JOE'.

AS: This keyword assigns an alias or a temporary name to a field or an expression in the field list. For example, AS f2_sub_up assigns the name f2_sub_up to the expression substring(upper('mr joe doe'), 4, 3).

You cannot do any of the following:

A) SELECT FROM TABLE dbtabl FIELDS Of1, upper(left('mr joe doe', 6)) AS f2_up_left, f3,...: This expression uses the wrong SQL function for strings to get the desired result. The left function returns the leftmost characters of a string with a specified length, ignoring the trailing blanks. For example, left('mr joe doe', 6) returns 'mr joe'. Applying the upper function to this result returns 'MR JOE', which is not the same as 'JOE'.

B) SELECT FROM TABLE dbtabl FIELDS Of1, left(lower(substring('mr joe doe', 4, 3)), 3) AS f2_left_lo_sub, f3,...: This expression uses unnecessary and incorrect SQL functions for strings to get the desired result. The lower function converts all uppercase characters in a string to lowercase. For example, lower(substring('mr joe doe', 4, 3)) returns 'joe'. Applying the left function to this result with the same length returns 'joe' again, which is not the same as 'JOE'.

D) SELECT FROM TABLE dbtabl FIELDS Of1, substring(lower(upper('mr joe doe')), 4, 3) AS f2_sub_lo_up, f3,...: This expression uses unnecessary and incorrect SQL functions for strings to get the desired result. The lower function converts all uppercase characters in a string to lowercase, and the upper function converts all lowercase characters in a string to uppercase. Applying both functions to the same string cancels out the effect of each other and returns the original string. For example, lower(upper('mr joe doe')) returns 'mr joe doe'. Applying the substring function to this result returns 'joe', which is not the same as 'JOE'.

To Get Premium Files for C_ABAPD_2309 Visit

https://www.p2pexams.com/products/c_abapd_2309

For More Free Questions Visit

https://www.p2pexams.com/sap/pdf/c-abapd-2309