C_ABAPD_2309 Q&As

SAP Certified Associate - Back-End Developer - ABAP Cloud

Pass SAP C_ABAPD_2309 Exam with 100% Guarantee

Free Download Real Questions & Answers **PDF** and **VCE** file from:

https://www.pass4itsure.com/c_abapd_2309.html

100% Passing Guarantee
100% Money Back Assurance

Following Questions and Answers are all new published by SAP Official Exam Center

QUESTION 1

You are given the following information:

```
1  SELECT SINGLE *
2  FROM SPFLI
3  WHERE CARRID = 'LH' AND CONNID = '1234'
4  INTO @data(ls)
```

1.

The data source "spfli" on line #2 is an SAP HANA database table

2.

"spfli" will be a large table with over one million rows.

3.

This program is the only one in the system that accesses the table.

4.

This program will run rarely.

Based on this information, which of the following general settings should you set for the spfli database table? Note: There are 2 correct answers to this question.

A. "Storage Type" to "Column Store"

B. "Load Unit to "Column Loadable"

C. "Storage Type" to "Row Store"

D. "Load Unit\\' to \\'Page Loadable"

Correct Answer: CD

Based on the given information, the spfli database table should have the following general settings: "Storage Type" to "Row Store": This setting determines how the data is stored in the SAP HANA database. Row store is suitable for tables that are accessed by primary key or by a small number of columns. Column store is suitable for tables that are accessed by a large number of columns or by complex analytical queries. Since the spfli table is a large table with over one million rows, and this program is the only one in the system that accesses the table, it is likely that the program will use primary key access or simple queries to access the table. Therefore, row store is a better choice than column store for this table12. "Load Unit" to "Page Loadable": This setting determines how the data is loaded into the memory when the table is accessed. Page loadable means that the data is loaded in pages of 16 KB each, and only the pages that are needed are loaded. Column loadable means that the data is loaded in columns, and only the columns that are needed are loaded. Since the spfli table is a row store table, and this program will run rarely, it is more efficient to use page loadable than column loadable for this table. Page loadable will reduce the memory consumption and the loading time of the table13. References: 1: Table Types in SAP HANA | SAP Help Portal 2: [Row Store vs Column Store in SAP HANA | SAP Blogs] 3: [Load Unit | SAP Help Portal]

QUESTION 2

Given the following code in an SAP S/4HANA Cloud private edition tenant:

```
1 CLASS zcl_demo_class DEFINITION.
2 METHODS: m1.
3 ENDCLASS..
4 CLASS zcl_demo_class_Implementation.
5 METHOD m1.
6 CALL FUNCTION 'ZF1'.
7 ENDMETHOD
8 ENDCLASS.
```

The class zcl_demo_class is in a software component with the language version set to "ABAP Cloud". The function module ZF1\\' is in a different software component with the language version set to "Standard ABAP". Both the class and function module are customer created.

Regarding line #6, which of the following are valid statements? Note: There are 2 correct answers to this question.

A. ZF1\\' can be called only if it is released for cloud development.

B. \\'ZF1\\' can be called if a wrapper is created for it and the wrapper itself is released for cloud development.

C. "ZF1" can be called whether it is released or not for cloud development

D. ZF1" can be called if a wrapper is created for it but the wrapper itself is not released for cloud development.

Correct Answer: AB

The ABAP Cloud Development Model requires that only public SAP APIs and extension points are used to access SAP functionality and data. These APIs and extension points are released by SAP and documented in the SAP API BusinessHub1. Customer-created function modules are not part of the public SAP APIs and are not released for cloud development. Therefore, calling a function module directly from an ABAP Cloud class is not allowed and will result in a syntax error. However, there are two possible ways to call a function module indirectly from an ABAP Cloud class: Create a wrapper class or interface for the function module and release it for cloud development. A wrapper is a class or interface that encapsulates the function module and exposes its functionality through public methods or attributes. The wrapper must be created in a software component with the language version set to "Standard ABAP" and must be marked as released for cloud development using the annotation @EndUserText.label. The wrapper can then be called from an ABAP Cloud class using the public methods or attributes2. Use the ABAP Cloud Connector to call the function module as a remote function call (RFC) from an ABAP Cloud class. The ABAP Cloud Connector is a service that enables the secure and reliable communication between SAP BTP, ABAP environment and on-premise systems. The function module must be exposed as an RFC-enabled function module in the on-premise system and must be registered in the ABAP Cloud Connector. The ABAP Cloud class can then use the class cl_rfc_destination_service to get the destination name and the class cl_abap_system to create a proxy object for the function module. The proxy object can then be used to call the function module3. References: 1: SAP API Business Hub 2: Creating an ABAP Cloud Project | SAP Help Portal 3: Calling Remote Function Modules | SAP Help Portal

QUESTION 3

Exhibit

(Sorry, we do not have a more clear image. If we have a better resource for the image, we will update this one immediately.)

Which of the following ABAP SQL snippets are syntactically correct ways to provide a value for the parameter on line #4? Note: There are 2 correct answers to this question

A. ...SELECT * FROM deno_cds_param_view_entity (p_date = @ (cl_abap_context_info- >get_system_date ())...

B. ...SELECT * FROM deno_cds_param_view_entity (p_date - \\'20230101\\')...)

C. ...SELECT * FROM demo_cds_param_view_entity (p_date: 20238181\\')...)

D. ...SELECT * FROM demo_cds_param_view entity (p_date: $session.system_date)...

Correct Answer: AB

QUESTION 4

Image:

(Sorry, we do not have a more clear image. If we have a better resource for the image, we will update this one immediately.) In the following ABAP SQL code, what are valid case distinctions? Note: There are 2 correct answers to this question.

☐ A.
```
SELECT FROM dbtab1 FIELDS F1,
CASE
WHEN F2 = '1' THEN 'Value 1
WHEN f2'2' THEN 'Value 2' ELSE "Value for the rest' END AS f case
INTO TABLE @et t1.
```

☐ B.
```
SELECT FROM dbtab1 FIELDS f1,
CASE f2
WHEN '1' THEN 'Value 1'
WHEN '2' THEN 'Value 2'
ELSE "Value for the rest' END AS f_case
INTO TABLE @gt_t1.
```

☐ C.
```
SELECT FROM dbtab1 FIELDS F1,
CASE f2,
WHEN '1' THEN 'Value 1',
WHEN '2' THEN 'Value 2',
WHEN OTHERS "Value for the rest", ENDCASE AS f_case
INTO TABLE @gt t1.
```

☐ D.
```
SELECT FROM dbtab1 FIELDS F1,
CASE
WHEN F2 = '1' THEN "Value 1' WHEN f2 < f3 AND f2 = '2' THEN "Value 2'
WHEN OTHERS 'Value for the rest' ENDCASE AS f_case
INTO TABLE @gt t1.
```

A. Option A

B. Option B

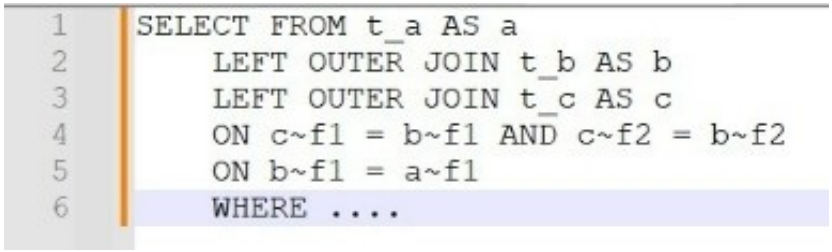
C. Option C

D. Option D

Correct Answer: AB

QUESTION 5

In this nested join below in which way is the join evaluated?



```
SELECT FROM t_a AS a
    LEFT OUTER JOIN t_b AS b
    LEFT OUTER JOIN t_c AS c
    ON c~f1 = b~f1 AND c~f2 = b~f2
    ON b~f1 = a~f1
    WHERE ....
```

A. From the left to the right in the order of the tables:

1.

a is joined with b

2.

b is joined with c

B. From the right to the left in the order of the tables:

1.

b is joined with c.

2.

b is joined with a.

C. From the top to the bottom in the order of the on conditions 1. b is joined with c

2.

a is joined with b

D. From the bottom to the top in the order of the on conditions:

1.

a is joined with b

2.

b is joined with c

Correct Answer: C

The nested join is evaluated from the top to the bottom in the order of the ON conditions. This means that the join expression is formed by assigning each ON condition to the directly preceding JOIN from left to right. The join expression can be parenthesized implicitly or explicitly to show the order of evaluation. In this case, the implicit parentheses are as follows: SELECT * FROM (a INNER JOIN (b INNER JOIN c ON b~c = c~c) ON a~b = b~b) This means that the first join expression is b INNER JOIN c ON b~c = c~c, which joins the columns of tables b and c based on the condition that b~c equals c~c. The second join expression is a INNER JOIN (b INNER JOIN c ON b~c = c~c) ON a~b = b~b, which joins the columns of table a and the result of the first join expression based on the condition that a~b equals b~b. The final result set contains all combinations of rows from tables a, b, and c that satisfy both join conditions. References: 1: SELECT, FROM JOIN - ABAP Keyword Documentation - SAP Online Help

QUESTION 6

What is the sequence priority when evaluating a logical expression?

A. NOT 1

B. OR 3

C. AND 2

D. A B C

E. CAB

F. A C B

G. B A C

Correct Answer: C

The sequence priority when evaluating a logical expression is C. A C B, which means NOT, AND, OR. This is the order of precedence of the Boolean operators in ABAP, which determines how the system implicitly parenthesizes all logical expressions that are not closed by explicit parentheses. The operator with the highest priority is evaluated first, and the operator with the lowest priority is evaluated last. The order of precedence of the Boolean operators in ABAP is as follows12: NOT: The NOT operator is a unary operator that negates the logical expression that follows it. It has the highest priority and is evaluated before any other operator. For example, in the expression NOT a AND b, the NOT operator is applied to a first, and then the AND operator is applied to the result and b. AND: The AND operator is a binary operator that returns true if both logical expressions on its left and right are true, and false otherwise. It has the second highest priority and is evaluated before the OR and EQUIV operators. For example, in the expression a AND b OR c, the AND operator is applied to a and b first, and then the OR operator is applied to the result and c. OR: The OR operator is a binary operator that returns true if either or both logical expressions on its left and right are true, and false otherwise. It has the third highest priority and is evaluated after the NOT and AND operators, but before the EQUIV operator. For example, in the expression a OR b EQUIV c, the OR operator is applied to a and b first, and then the EQUIV operator is applied to the result and c. EQUIV: The EQUIV operator is a binary operator that returns true if both logical expressions on its left and right have the same truth value, and false otherwise. It has the lowest priority and is evaluated after all other operators. For example, in the expression a AND b EQUIV c OR d, the EQUIV operator is applied to a AND b and c last, after the AND and OR operators are applied. References: 1: log_exp - Boolean Operators and Parentheses - ABAP Keyword Documentation SAP Online Help 2: Logical Expressions (log_exp) - ABAP Keyword Documentation - SAP Online Help

QUESTION 7

Exhibit:

```
1  INTERFACE if1.
2      METHODS m1.
3  ENDINTERFACE.
4
5  CLASS cl1 DEFINITION.
6  PUBLIC SECTION.
7      INTERFACES if1.
8      METHODS m2.
9  ENDCLASS.
10 ...
11 *In a method of another class
12 DATA go_if1 TYPE REF TO if1.
13 DATA go_cl1 TYPE REF TO cl1.
14 go_cl1 = NEW #(...).
15 go_if1 = go_cl1.
```

What are valid statements? Note: There are 3 correct answers to this question.

A. go_if 1 may call method ml with go_ift->ml().

B. Instead of go ell = NEW #(...) you could use go ifl = NEW cll(. ...).

C. go_cll may call method ml with go_dl->ifl-ml().

D. Instead of go_cll = NEW #() you could use go_iff - NEW #(...).

E. go_ifl may call method m2 with go if->m2(...).

Correct Answer: ABE

The following are the explanations for each statement:

A: This statement is valid. go_ifl may call method ml with go_ifl->ml(). This is because go_ifl is a data object of type REF TO ifl, which is a reference to the interface ifl. The interface ifl defines a method ml, which can be called using the reference variable go_ifl. The class cll implements the interface ifl, which means that it provides an implementation of the method ml. The data object go_ifl is assigned to a new instance of the class cll using the NEW operator and the inline declaration operator @DATA. Therefore, when go_ifl->ml() is called, the implementation of the method ml in the class cll is executed123

B: This statement is valid. Instead of go_cll = NEW #(...) you could use go_ifl = NEW cll(...). This is because go_ifl is a data object of type REF TO ifl, which is a reference to the interface ifl. The class cll implements the interface ifl, which means that it is compatible with the interface ifl. Therefore, go_ifl can be assigned to a new instance of the class cll using the NEW operator and the class name cll. The inline declaration operator @DATA is optional in this case, as go_ifl is already declared. The parentheses after the class name cll can be used to pass parameters to the constructor of the class cll, if any123

E: This statement is valid. go_ifl may call method m2 with go_ifl->m2(...). This is because go_ifl is a data object of type REF TO ifl, which is a reference to the interface ifl. The class cll implements the interface ifl, which means that it inherits

all the components of the interface ifl. The class cll also defines amethod m2, which is a public method of the class cll. Therefore, go_ifl can call the method m2 using the reference variable go_ifl. The method m2 is not defined in the interface ifl, but it is accessible through the interface ifl, as the interface ifl is implemented by the class cll. The parentheses after the method name m2 can be used to pass parameters to the method m2, if any123 The other statements are not valid, as they have syntax errors or logical errors. These statements are:

C: This statement is not valid. go_cll may call method ml with go_cll->ifl~ml(). This is because go_cll is a data object of type REF TO cll, which is a reference to the class cll. The class cll implements the interface ifl, which means that it inherits all the components of the interface ifl. The interface ifl defines a method ml, which can be called using the reference variable go_cll. However, the syntax for calling an interface method using a class reference is go_cll->ml(), not go_cll->ifl~ml (). The interface component selector ~ is only used when calling an interface method using an interface reference, such as go_ifl->ifl~ml(). Using the interface component selector ~ with a class reference will cause a syntax error123

D: This statement is not valid. Instead of go_cll = NEW #() you could use go_ifl = NEW #(...). This is because go_ifl is a data object of type REF TO ifl, which is a reference to the interface ifl. The interface ifl cannot be instantiated, as it does not have an implementation. Therefore, go_ifl cannot be assigned to a new instance of the interface ifl using the NEW operator and the inline declaration operator @DATA. This will cause a syntax error or a runtime error. To instantiate an interface, you need to use a class that implements the interface, such as the class cll123 References: INTERFACES - ABAP Keyword Documentation, CLASS - ABAP Keyword Documentation, NEW - ABAP Keyword Documentation

---

QUESTION 8

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

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.

Correct Answer: ABD

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. Secondarykeys 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. References: 1:

Secondary Table Key - ABAP Keyword Documentation 2: READ TABLE - ABAP Keyword Documentation 3: Sorted Tables - ABAP Keyword Documentation :

Hashed Tables - ABAP Keyword Documentation

QUESTION 9

Which of the following are incomplete ABAP types? Note: There are 2 correct answers to this question.

A. String

B. T

C. C

D. P

Correct Answer: CD

Explanation: Incomplete ABAP types are types that do not specify all the attributes of a data type, such as the length, the number of decimal places, or the value range. Incomplete types can only be used for the typing of field symbols and

formal parameters, not for the definition of data objects or constants. Incomplete types can be either predefined or user-defined1.

The following are incomplete ABAP types:

C. C is a type for character strings with a generic length. The length of the character string has to be specified when a data object or a constant is defined with this type. For example, DATA text TYPE c LENGTH 10 defines a data object named text with a type c and a length of 10 characters2. D. P is a type for packed numbers with a generic length and a generic number of decimal places. The length and the number of decimal places of the packed number have to be specified when a data object or a constant is defined with this type. For example, DATA amount TYPE p LENGTH 8 DECIMALS 2 defines a data object named amount with a type p, a length of 8 bytes, and 2 decimal places3. The following are not incomplete ABAP types, because they specify all the attributes of a data type:

A. String is a type for variable-length character strings. The length of the character string is determined at runtime and can vary from 0 to 2,147,483,647 characters. The length does not have to be specified when a data object or a constant is defined with this type. For example, DATA text TYPE string defines a data object named text with a type string and a variable length4. B. T is a type for time values in the format HHMMSS. The length of the time value is fixed at 6 characters and does not have to be specified when a data object or a constant is defined with this type. For example, DATA time TYPE t defines a data object named time with a type t and a length of 6 characters. References: 1: Generic ABAP Types ABAP Keyword Documentation 2: C - ABAP Keyword Documentation 3: P - ABAP Keyword Documentation 4: String - ABAP Keyword Documentation : T - ABAP Keyword Documentation

QUESTION 10

Class super has subclass sub. Which rules are valid for the sub constructor? Note: There are 2 correct answers to this question.

A. The method signature can be changed.

B. Import parameters can only be evaluated after calling the constructor of super.

C. The constructor of super must be called before using any components of your own instance.

D. Events of your own instance cannot be raised before the registration of a handler in super.

Correct Answer: AC

The sub constructor is the instance constructor of the subclass sub that inherits from the superclass super. The sub constructor has some rules that it must follow when it is defined and implemented12. Some of the valid rules are:

The method signature can be changed: This is true. The sub constructor can have a different method signature than the super constructor, which means that it can have different input parameters, output parameters, or exceptions. However,

the sub constructor must still call the super constructor with appropriate actual parameters that match its interface12.

The constructor of super must be called before using any components of your own instance: This is true. The sub constructor must ensure that the super constructor is called explicitly using super->constructor before accessing any instance

components of its own class, such as attributes or methods. This is because the super constructor initializes the inherited components of the subclass and sets the self-reference me-> to the current instance12.

You cannot do any of the following:

Import parameters can only be evaluated after calling the constructor of super:

This is false. The sub constructor can evaluate its own import parameters before calling the constructor of super, as long as it does not access any instance components of its own class. For example, the sub constructor can use its import

parameters to calculate some values or check some conditions that are needed for calling the super constructor12.

Events of your own instance cannot be raised before the registration of a handler in super: This is false. The sub constructor can raise events of its own instance before calling the constructor of super, as long as it does not access any

instance components of its own class. For example, the sub constructor can raise an event to notify the consumers of the subclass about some status or error that occurred during the initialization of the subclass12.

References: 1: Inheritance and Constructors - ABAP Keyword Documentation - SAP Online Help 2: Using Static and Instance constructor methods | SAP Blogs

QUESTION 11

Which internal table type allows unique and non-unique keys?

A. Sorted

B. Hashed

C. Standard

Correct Answer: C

The internal table type that allows both unique and non-unique keys is the standard table. A standard table has an internal linear index that can be used to access the table entries. The key of a standard table is always non-unique, which means that the table can contain duplicate entries. However, the system does not check the uniqueness of the key when inserting new entries, so the programmer can ensure that the key is unique by using appropriate logic. A standard table can be accessed either by using the table index or the key, but the response time for key access is proportional to the table size. The other two internal table types, sorted and hashed, do not allow non-unique keys. A sorted table is filled in sorted order according to the defined table key, which must be unique. A sorted table can be accessed either by using the table index or the key, but the response time for key access is logarithmically proportional to the table size. A hashed table can only be accessed by using a unique key, which must be specified when declaring the table. A hashed table has no index, and the response time for key access is constant, regardless of the table size. References: Internal Tables - ABAP Keyword Documentation, SAP ABAP: Types Of Internal Table Declaration - dan852.com

QUESTION 12

Which of the following are features of Core Data Services? Note: There are 3 correct answers to this question.

A. Inheritance

B. Associations

C. Annotations

D. Delegation

E. Structured Query Language (SQL)

Correct Answer: BCE

Core Data Services (CDS) is a framework for defining and consuming semantically rich data models in SAP HANA. CDS supports various features that enhance the capabilities of SQL and enable developers to create data models that are optimized for performance, readability, and extensibility12. Some of the features of CDS are: Associations: Associations are a way of defining relationships between CDS entities, such as tables or views. Associations enable navigation and path expressions in CDS queries, which allow accessing data from related entities without explicit joins. Associations also support cardinality, referential constraints, and cascading options34. Annotations: Annotations are a way of adding metadata to CDS entities or their elements, such as fields or parameters. Annotations provide additionalinformation or instructions for the CDS compiler, the database, or the consumers of the CDS views. Annotations can be used for various purposes, such as defining access control, UI rendering, OData exposure, or search capabilities5 . Structured Query Language (SQL): SQL is the standard language for querying and manipulating data in relational databases. CDS is based on SQL and extends it with additional features and syntax. CDS supports SQL features such as joins, aggregations, filters, expressions, functions, and subqueries. CDS also supports SQL Script, which is a scripting language for stored procedures and functions in SAP HANA . You cannot do any of the following: Inheritance: Inheritance is not a feature of CDS. Inheritance is a concept in object- oriented programming that allows a class to inherit the properties and methods of another class. CDS does not support object-oriented programming or classes. Delegation: Delegation is not a feature of CDS. Delegation is a concept in object- oriented programming that allows an object to delegate some of its responsibilities to another object. CDS does not support object-oriented programming or objects. References: 1: Core Data Services (CDS) | CAPire 2: Core Data Services [CDS] in SAP S/4 HANA | SAP Blogs 3: Associations in Core Data Services (CDS) | SAP Help Portal 4: [CDS DDL - Association - ABAP Keyword Documentation - SAP Online Help] 5: [Annotations in Core Data Services (CDS) | SAP Help Portal] : [CDS DDL - Annotation - ABAP Keyword Documentation - SAP Online Help] : [Structured Query Language (SQL) | SAP Help Portal] : [CDS DDL - SQL Features - ABAP Keyword Documentation - SAP Online Help] : [Object-Oriented Programming in ABAP | SAP Help Portal]

QUESTION 13

In ABAP SQL, which of the following retrieves the association field_Airline-Name of a CDS view?

A. _Airline-Name

B. /_Airline Name

C. @_Airline-Name

D. "_Airline Name

Correct Answer: C

In ABAP SQL, the syntax to retrieve the association field of a CDS view is to use the @ sign followed by the association

name and the field name, separated by a period sign (.). For example, to retrieve the association field _Airline-Name of a CDS view, the syntax is @_Airline.Name. This syntax allows the access to the fields of the target data source of the association without explicitly joining the data sources1. The other options are incorrect because they use the wrong symbols or formats to access the association field. References: 1: Path Expressions - ABAP Keyword Documentation

QUESTION 14

When processing a loop with the statement DO... ENDDO, what system variable contains the implicit loop counter?

A. sy-linno

B. sy-labix

C. sy-subrc

D. sy-index

Correct Answer: D

Explanation: When processing a loop with the statement DO... ENDDO, the system variable that contains the implicit loop counter is sy-index. The loop counter is a numeric value that indicates how many times the loop has been executed.

The loop counter is initialized to 1 before the first execution of the loop and is incremented by 1 after each execution. The loop counter can be used to control the number of loop iterations or to access the loop elements by index. The loop

counter can also be accessed or modified within the loop body, but this is not recommended as it may cause unexpected results or errors1.

For example, the following code snippet uses the loop counter sy-index to display the numbers from 1 to 10:

DO 10 TIMES. WRITE: / sy-index. ENDDO.

The output of this code is:

1 2 3 4 5 6 7 8 9 10

References: 1: DO - ABAP Keyword Documentation

QUESTION 15

```
1  Given this code,
2  DATA: go_super TYPE REF TO lcl_super,
3        go_sub1  TYPE REF TO lcl_sub1.
4  ...
5  go_sub1 = CAST #( go_super ).
6  go_sub1->sub1_meth1(...).
7
```

Which predicate condition can you ensure that the CAST will work?

A. IS SUPPLIED

B. IS NOT INITIAL

C. IS INSTANCE OF

D. IS BOUND

Correct Answer: C

The predicate condition that can be used to ensure that the CAST will work is IS INSTANCE OF. The IS INSTANCE OF predicate condition checks whether the operand is an instance of the specified class or interface. This is useful when you want to perform a downcast, which is a conversion from a more general type to a more specific type. A downcast can fail if the operand is not an instance of the target type, and this can cause a runtime error. Therefore, you can use the IS INSTANCE OF predicate condition to check whether the downcast is possible before using the CAST operator12. For example: The following code snippet uses the IS INSTANCE OF predicate condition to check whether the variable g_super is an instance of the class lcl_super. If it is, the CAST will work and the variable g_sub1 will be assigned the value of g_super. DATA: g_super TYPE REF TO lcl_super, g_sub1 TYPE REF TO lcl_sub1. IF g_super IS INSTANCE OF lcl_super. g_sub1 = CAST #(g_super). g_sub1->method(...). ENDIF. You cannot do any of the following: IS SUPPLIED: The IS SUPPLIED predicate condition checks whether an optional parameter of a method or a function module has been supplied by the caller. This is useful when you want to handle different cases depending onwhether the parameter has a value or not. However, this predicate condition has nothing to do with the CAST operator or the type of the operand12. IS NOT INITIAL: The IS NOT INITIAL predicate condition checks whether the operand has a non-initial value. This is useful when you want to check whether the operand has been assigned a value or not. However, this predicate condition does not guarantee that the CAST will work, because the operand may have a value but not be an instance of the target type12. IS BOUND: The IS BOUND predicate condition checks whether the operand is a bound reference variable. This is useful when you want to check whether the operand points to an existing object or not. However, this predicate condition does not guarantee that the CAST will work, because the operand may point to an object but not be an instance of the target type12. References: 1: Predicate Expressions - ABAP Keyword Documentation - SAP Online Help 2: ABAP - Predicates | SAP Community