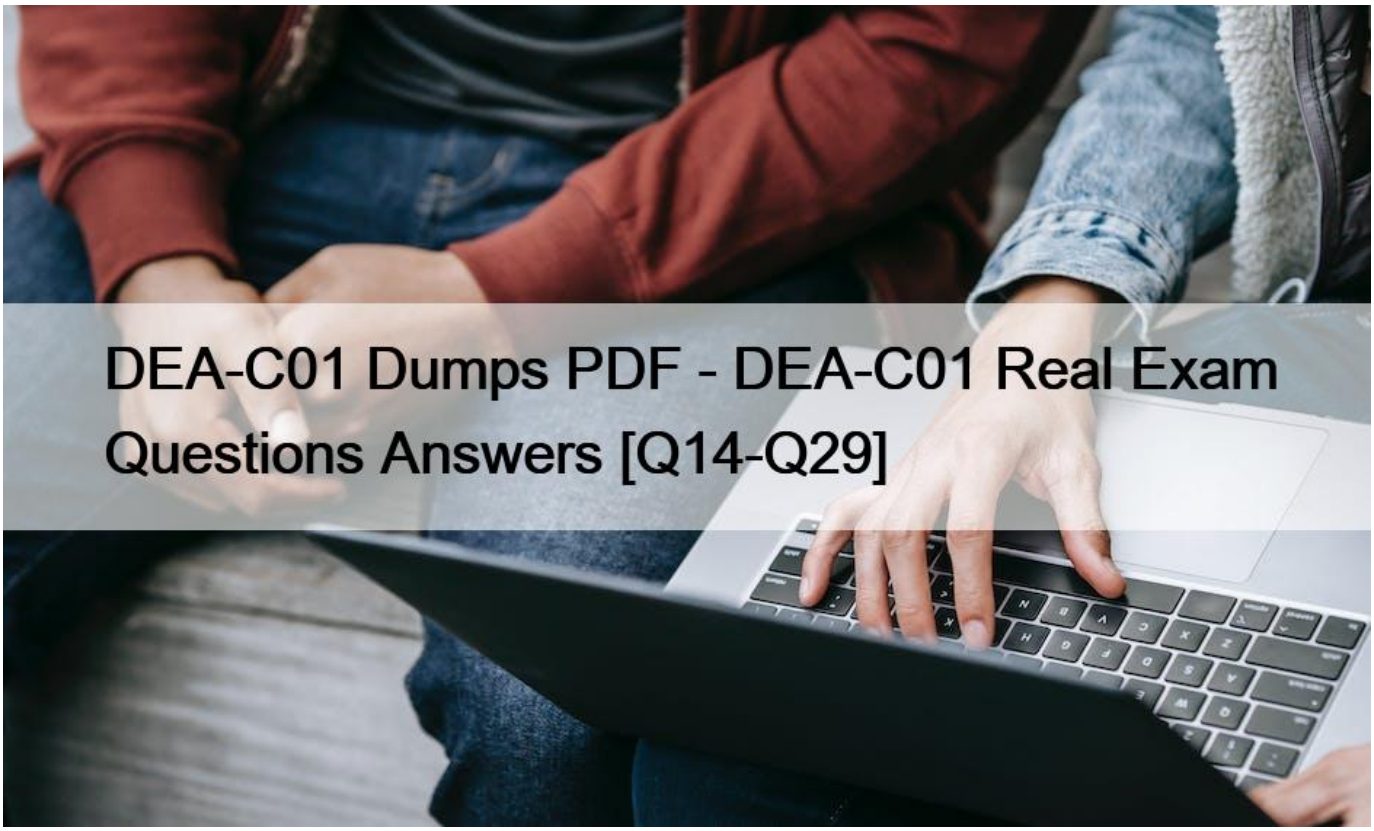
DEA-C01 Dumps PDF - DEA-C01 Real Exam Questions Answers [Q14-Q29



DEA-C01 Dumps PDF - DEA-C01 Real Exam Questions Answers
Get Started: DEA-C01 Exam [year] Dumps Snowflake PDF Questions

NO.14 A SQL UDF evaluates an arbitrary SQL expression and returns the result(s) of the expression. Which value type it can returns?
* Single Value
* A Set of Rows
* Scaler or Tabular depend on input SQL expression
* Regex

NO.15 While running an external function, me following error message is received:

Error:function received the wrong number of rows

What iscausing this to occur?
* External functions do not support multiple rows
* Nested arrays are not supported in the JSON response
* The JSON returned by the remote service is not constructed correctly
* The return message did not produce the same number of rows that it received

Explanation

The error message “function received the wrong number of rows” is caused by the return message not producing the same number of rows that it received. External functions require that the remote service returns exactly one row for each input row that it receives from Snowflake. If the remote service returns more or fewer rows than expected, Snowflake will raise an error and abort the function execution. The other options are not causes of this error message. Option A is incorrect because external functions do support multiple rows as long as they match the input rows. Option B is incorrect because nested arrays are supported in the JSON response as long as they conform to the return type definition of the external function. Option C is incorrect because the JSON returned by the remote service may be constructed correctly but still produce a different number of rows than expected.

NO.16 A Data Engineer needs to ingest invoice data in PDF format into Snowflake so that the data can be queried and used in a forecasting solution.

….. recommended way to ingest this data?
* Use Snowpipe to ingest the files that land in an external stage into a Snowflake table
* Use a COPY INTO command to ingest the PDF files in an external stage into a Snowflake table with a VARIANT column.
* Create an external table on the PDF files that are stored in a stage and parse the data nto structured data
* Create a Java User-Defined Function (UDF) that leverages Java-based PDF parser libraries to parse PDF data into structured data
Explanation

The recommended way to ingest invoice data in PDF format into Snowflake is to create a Java User-Defined Function (UDF) that leverages Java-based PDF parser libraries to parse PDF data into structured data. This option allows for more flexibility and control over how the PDF data is extracted and transformed. The other options are not suitable for ingesting PDF data into Snowflake. Option A and B are incorrect because Snowpipe and COPY INTO commands can only ingest files that are in supported file formats, such as CSV, JSON, XML, etc. PDF files are not supported by Snowflake and will cause errors or unexpected results.

Option C is incorrect because external tables can only query files that are in supported file formats as well.

PDF files cannot be parsed by external tables and will cause errors or unexpected results.

NO.17 How Data Engineer can do Monitoring of Files which are Staged Internally during Continuous data pipelines loading process? [Select all that apply]
* She Can Monitor the files using Metadata maintained by Snowflake i.e. file-name,last_modified date etc.
* Snowflake retains historical data for COPY INTO commands executed within the pre-vious 14 days.
* She can Monitor the status of each COPY INTO <table> command on the History tab page of the classic web interface.
* She can use the DATA_LOAD_HISTORY Information Schema view to retrieve the history of data loaded into tables using the COPY INTO command.
* She can use the DATA_VALIDATE function to validate the data files She have loaded and can retrieve any errors encountered during the load.
Explanation

Monitoring Files Staged Internally

Snowflake maintains detailed metadata for each file uploaded into internal stage (for users, tables, and stages), including:

File name

File size (compressed, if compression was specified during upload)

LAST_MODIFIED date, i.e. the timestamp when the data file was initially staged or when it was last modified, whichever is later In addition, Snowflake retains historical data for COPY INTO commands executed within the pre-vious 14 days. The metadata can be used to monitor and manage the loading process, including de-leting files after upload completes:

Use the LIST command to view the status of data files that have been staged.

Monitor the status of each COPY INTO <table> command on the History tab page of the classic web interface.

Use the VALIDATE function to validate the data files you’ve loaded and retrieve any errors en-countered during the load.

Use the LOAD_HISTORY Information Schema view to retrieve the history of data loaded into tables using the COPY INTO command.

NO.18 When would a Data engineer use table with the flatten function instead of the lateral flatten combination?
* When TABLE with FLATTENrequires another source in the from clause to refer to
* WhenTABLE with FLATTENrequires no additional source m the from clause to refer to
* Whenthe LATERALFLATTENcombination requires no other source m the from clause to refer to
* When table withFLATTENis acting like a sub-query executed for each returned row

Explanation

The TABLE function with the FLATTEN function is used to flatten semi-structured data, such as JSON or XML, into a relational format. The TABLE function returns a table expression that can be used in the FROM clause of a query. The TABLE function with the FLATTEN function requires another source in the FROM clause to refer to, such as a table, view, or subquery that contains the semi-structured data. For example:

SELECT t.value:city::string AS city, f.value AS population FROM cities t, TABLE(FLATTEN(input => t.value:population)) f; In this example, the TABLE function with the FLATTEN function refers to the cities table in the FROM clause, which contains JSON data in a variant column named value. The FLATTEN function flattens the population array within each JSON object and returns a table expression with two columns: key and value.

The query then selects the city and population values from the table expression.

NO.19 Which two Account usage views can be used for auditing Dynamic data masking purpose?
* MASKING POLICIES
* POLICY_REFERENCES
* DYNAMIC MASKING POLICIES
* DYNAMIC POLICY_REFERENCES

NO.20 Elon, a Data Engineer, needs to Split Semi-structured Elements from the Source files and load them as an array into Separate Columns.

Source File:

1.+——————————————————————————————-+

2.| $1 |

3.|—————————————————————————————-|

4.| {“mac_address”: {“host1”: “197.128.1.1”,”host2″: “197.168.0.1”}}, |

5.| {“mac_address”: {“host1”: “197.168.2.1”,”host2″: “197.168.3.1”}} |

6.+————————————————————————————————-+ Output: Splitting the Machine Address as below.

1.COL1 | COL2 |

2.|———-+———-|

3.| [| [|

4.| “197”, | “197”, |

5.| “128”, | “168”, |

6.| “1”, | “0”, |

7.| “1” | “1” |

8.|] |] |

9.| [| [|

10.| “197”, | “197”, |

11.| “168”, | “168”, |

12.| “2”, | “3”, |

13.| “1” | “1” |

14.|] |]

Which SnowFlake Function can Elon use to transform this semi structured data in the output for-mat?
* CONVERT_TO_ARRAY
* SPLIT
* GROUP_BY_CONNECT
* NEST

NO.21 Mark a Data Engineer, looking to implement streams on local views & want to use change tracking metadata for one of its Data Loading use case. Please select the incorrect understanding points of Mark with respect to usage of Streams on Views?
* For streams on views, change tracking must be enabled explicitly for the view and un-derlying tables to add the hidden columns to these tables.
* The CDC records returned when querying a stream rely on a combination of the offset stored in the stream and the change tracking metadata stored in the table.
* Views with GROUP BY & LIMIT Clause are supported by Snowflake.
* As an alternative to streams, Snowflake supports querying change tracking metadata for views using the CHANGES clause for SELECT statements.

* Enabling change tracking adds a pair of hidden columns to the table and begins storing change tracking metadata. The values in these hidden CDC data columns provide the input for the stream metadata columns. The columns consume a small amount of stor-age.
Explanation

A stream object records data manipulation language (DML) changes made to tables, including in-serts, updates, and deletes, as well as metadata about each change, so that actions can be taken us-ing the changed data. This process is referred to as change data capture (CDC). An individual table stream tracks the changes made to rows in a source table. A table stream (also referred to as simply a “stream”) makes a “change table” available of what changed, at the row level, between two transac-tional points of time in a table. This allows querying and consuming a sequence of change records in a transactional fashion.

Streams can be created to query change data on the following objects:

Standard tables, including shared tables.

Views, including secure views

Directory tables

External tables

When created, a stream logically takes an initial snapshot of every row in the source object (e.g. ta-ble, external table, or the underlying tables for a view) by initializing a point in time (called an off-set) as the current transactional version of the object. The change tracking system utilized by the stream then records information about the DML changes after this snapshot was taken. Change rec-ords provide the state of a row before and after the change. Change information mirrors the column structure of the tracked source object and includes additional metadata columns that describe each change event.

Note that a stream itself does not contain any table data. A stream only stores an offset for the source object and returns CDC records by leveraging the versioning history for the source object. When the first stream for a table is created, a pair of hidden columns are added to the source table and begin storing change tracking metadata. These columns consume a small amount of storage. The CDC records returned when querying a stream rely on a combination of the offset stored in the stream and the change tracking metadata stored in the table. Note that for streams on views, change tracking must be enabled explicitly for the view and underlying tables to add the hidden columns to these tables.

Streams on views support both local views and views shared using Snowflake Secure Data Sharing, including secure views. Currently, streams cannot track changes in materialized views.

Views with the following operations are not yet supported:

GROUP BY clauses

QUALIFY clauses

Subqueries not in the FROM clause

Correlated subqueries

LIMIT clauses

Change Tracking:

Change tracking must be enabled in the underlying tables.

Prior to creating a stream on a view, you must enable change tracking on the underlying tables for the view.

Set the CHANGE_TRACKING parameter when creating a view (using CREATE VIEW) or later (using ALTER VIEW).

As an alternative to streams, Snowflake supports querying change tracking metadata for tables or views using the CHANGES clause for SELECT statements. The CHANGES clause enables query-ing change tracking metadata between two points in time without having to create a stream with an explicit transactional offset.

NO.22 UDTFs also called a table function, returns zero, one, or multiple rows for each input row?
* YES
* NO
Explanation

UDFs may be scalar or tabular.

A scalar function returns one output row for each input row. The returned row consists of a single column/value.

A tabular function, also called a table function, returns zero, one, or multiple rows for each input row. A tabular UDF is defined by specifying a return clause that contains the TABLE keyword and specifies the names and data types of the columns in the table results. Tabular UDFs are often called UDTFs (user-defined table functions) or table UDFs.

NO.23 When using the CURRENT_ROLE and CURRENT_USER functions with secure views that will be shared to other Snowflake accounts, Snowflake returns a NULL value for these functions?
* FALSE
* TRUE
Explanation

When using the CURRENT_ROLE and CURRENT_USER functions with secure views that will be shared to other Snowflake accounts, Snowflake returns a NULL value for these functions. The reason is that the owner of the data being shared does not typically control the users or roles in the account with which the view is being shared.

NO.24 Select the Correct statements with regard to using Federated authentication/SSO?
* Snowflake supports using MFA in conjunction with SSO to provide additional levels of security.
* Snowflake supports multiple audience values (i.e. Audience or Audience Restriction Fields) in the SAML 2.0 assertion from the identity provider to Snowflake.
* Snowflake supports SSO with Private Connectivity to the Snowflake Service for Snow-flake accounts on Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform.
* Snowflake supports using SSO with organizations, and you can use the corresponding URL in the SAML2 security integration.

NO.25 A Data Engineer is building a pipeline to transform a 1 TD tab e by joining it with supplemental tables The Engineer is applying filters and several aggregations leveraging Common TableExpressions (CTEs) using a size Medium virtual warehouse in a single query in Snowflake.

After checking the Query Profile, what is the recommended approach to MAXIMIZE performance of this query if the Profile shows data spillage?
* Enable clustering on the table
* Increase the warehouse size

* Rewrite the query to remove the CTEs.
* Switch to a multi-cluster virtual warehouse

Explanation

The recommended approach to maximize performance of this query if the Profile shows data spillage is to increase the warehouse size. Data spillage occurs when the query requires more memory than the warehouse can provide and has to spill some intermediate results to disk. This can degrade the query performance by increasing the disk IO time. Increasing the warehouse size can increase the amount of memory available for the query and reduce or eliminate data spillage.

NO.26 What are characteristics of Snowpark Python packages? (Select THREE).

Third-party packages can be registered as a dependency to the Snowpark session using the session, import () method.
* Python packages can access any external endpoints
* Python packages can only be loaded in a local environment
* Third-party supported Python packages are locked down to prevent hitting
* The SQL command DESCRIBE FUNCTION will list the imported Python packages of the Python User-Defined Function (UDF).
* Querying information__schema .packages will provide a list of supported Python packages and versions

Explanation

The characteristics of Snowpark Python packages are:

Third-party packages can be registered as a dependency to the Snowpark session using the session.import() method.

The SQL command DESCRIBE FUNCTION will list the imported Python packages of the Python User-Defined Function (UDF).

Querying information_schema.packages will provide a list of supported Python packages and versions.

These characteristics indicate how Snowpark Python packages can be imported, inspected, and verified in Snowflake. The other options are not characteristics of Snowpark Python packages. Option B is incorrect because Python packages can be loaded in both local and remote environments using Snowpark. Option C is incorrect because third-party supported Python packages are not locked down to prevent hitting external endpoints, but rather restricted by network policies and security settings.

NO.27 Mark the incorrect statement in case Data engineer using the COPY INTO <table> command to load data from files into Snowflake tables?
* For Data loading of files with semi-structured file formats (JSON, Avro, etc.), the only supported character set is UTF-16.
* For loading data from all semi-structured supported file formats (JSON, Avro, etc.), as well as unloading data, UTF-8 is the only supported character set.
* For Local environment, Files are first copied (“staged”) to an internal (Snowflake) stage, then loaded into a table.
* UTF-32 & UTF-16 both encoding character sets supported for loading data from de-limited files (CSV, TSV, etc.)

Explanation

For Data Loading of delimited files (CSV, TSV, etc.), the default character set is UTF-8. To use any other characters sets, you must explicitly specify the encoding to use for loading.

For semi-structured file formats (JSON, Avro, etc.), the only supported character set is UTF-8.

Rest of the statements are correct.

NO.28 In one of your created Schema, you have been required to create Internal Stages, what are the In-correct considerations you can noticed from the below options? [Select All that Apply]

* User stages can be altered or dropped just like Table Stage.
* Table stage type is designed to store files that are staged and managed by one or more users but only loaded into a single table.
* A named internal stage type can store files that are staged and managed by one or more users and loaded into one or more tables.
* A table stage is available for each table created in Snowflake.

Explanation

A stage specifies where data files are stored (i.e. “staged”) so that the data in the files can be loaded into a table.

Types of Internal Stages

User Stages

Table Stages

Named Stages

By default, each user and table in Snowflake is automatically allocated an internal stage for staging data files to be loaded. In addition, you can create named internal stages.

File staging information is required during both steps in the data loading process:

You must specify an internal stage in the PUT command when uploading files to Snowflake.

You must specify the same stage in the COPY INTO <table> command when loading data into a table from the staged files.

Consider the best type of stage for specific data files. Each option provides benefits and potential drawbacks.

User Stages

Each user has a Snowflake stage allocated to them by default for storing files. This stage is a con-venient option if your files will only be accessed by a single user, but need to be copied into multi-ple tables.

User stages have the following characteristics and limitations:

User stages are referenced using @~; e.g. use LIST @~ to list the files in a user stage.

Unlike named stages, user stages cannot be altered or dropped.

User stages do not support setting file format options. Instead, you must specify file format and copy options as part of the COPY INTO <table> command.

This option is not appropriate if:

Multiple users require access to the files.

The current user does not have INSERT privileges on the tables the data will be loaded into.

Table Stages

Each table has a Snowflake stage allocated to it by default for storing files. This stage is a conven-ient option if your files need to be

accessible to multiple users and only need to be copied into a sin-gle table.

Table stages have the following characteristics and limitations:

Table stages have the same name as the table; e.g. a table named mytable has a stage referenced as

@%mytable.

Unlike named stages, table stages cannot be altered or dropped.

Table stages do not support transforming data while loading it (i.e. using a query as the source for the COPY command).

Note that a table stage is not a separate database object; rather, it is an implicit stage tied to the table itself. A table stage has no grantable privileges of its own. To stage files to a table stage, list the files, query them on the stage, or drop them, you must be the table owner (have the role with the OWNERSHIP privilege on the table).

This option is not appropriate if you need to copy the data in the files into multiple tables.

Named Stages

Named stages are database objects that provide the greatest degree of flexibility for data loading:

Users with the appropriate privileges on the stage can load data into any table.

Because the stage is a database object, the security/access rules that apply to all objects apply. The privileges to use a stage can be granted or revoked from roles. In addition, ownership of the stage can be transferred to another role.

If you plan to stage data files that will be loaded only by you, or will be loaded only into a single table, then you may prefer to simply use either your user stage or the stage for the table into which you will be loading data.

Named stages are optional but recommended when you plan regular data loads that could involve multiple users and/or tables.

NO.29 Data Engineer Loading File named snowdata.tsv in the /datadir directory from his local machine to Snowflake stage and try to prefix the file with a folder named tablestage, please mark the correct command which helps him to load the files data into snowflake internal Table stage?
* put file://c:datadirsnowdata.tsv @~/tablestage;
* put file://c:datadirsnowdata.tsv @%tablestage;
* put file://c:datadirsnowdata.tsv @tablestage;
* put file:///datadir/snowdata.tsv @%tablestage;

Explanation

Execute PUT to upload (stage) local data files into an internal stage.

@% character combination identifies a table stage.

DEA-C01 Premium Exam Engine pdf Download: https://www.dumpsmaterials.com/DEA-C01-real-torrent.html]