Additional features for DataSync for Snowflake

To enhance your DataSync integration for **Snowflake**, you can configure the Snowflake Bulk Load Meshlet with the additional features available below!

What's on this page?

Add custom fields to Snowflake

There are currently 3 types of custom columns supported: IO insert columns, IO update columns, and a string field populated with the ServiceNow instance key. IO insert fields will be populated with the timestamp of the record insert into Snowflake, and IO update fields will be populated with the timestamp of the record insert into Snowflake and updated subsequently on record update into Snowflake.

To add your custom column names in your Snowflake tables, you can configure the **databaseConfig.json** file by following these steps:

1. Navigate to the directory where you saved your meshlet when installing, and open the databaseConfig.json file.

2. In the databaseConfig.json, the IO columns (insert, update) are grouped together. Each set of columns is a list.

   To add your custom column in the IO insert columns, IO update columns, and the key columns, add them in between the brackets:

```
{...
        "IODateTime": {
                "insert":[],
                "updated":[]
        },
        "key":[]
}
```

```
{...
        "IODateTime": {
                "insert":
["psp_example_insert"],
                "updated":
["psp_example_update"]
        },
        "key":["psp_example_key"]
}
```

   To have no custom columns, leave the lists empty.

```
{
   "SQLSyntax":{
      "showTables":"SHOW TABLES LIKE %s",
      "selectAll":"SELECT * FROM %s",
      "addColumn":"ALTER TABLE %s ADD COLUMN",
      "primaryKey":"Primary Key(%s)"
   },
   "tableDefinition":{
      "primaryKeys":[
      "sys_id"
      ]
   },
   "types":{
      "integer":"BIGINT",
      "boolean":"BOOLEAN",
      "glide_date":"DATE",
      "glide_date_time":"TIMESTAMP_LTZ(9)",
      "float":"DOUBLE",
      "reference":"VARCHAR",
      "default":"VARCHAR",
      "IODateTime": "TIMESTAMP_LTZ(9)",
      "key": "VARCHAR",
      "delete": "BOOLEAN"
   },
   "IODateTime":{
      "insert":["psp_example_insert"],
      "update":["psp_example_update"]
   },
   "key": ["psp_example_key"]
}
```

3

If you want the IO update column to be NULL on record inserts, set the following in your **application.yml**:

```
perspectium:
        snowflake:
                updateTimeOnInsert: true
```

By default, updateTimeOnInsert is true, and IO update columns are populated on record inserts.

By default, IO columns will behave as they do in ServiceNow; for rows inserted into Snowflake tables with a custom IO update column, the IO update column will be populated with the insert timestamp. This setting can be turned off with updateTimeOnInsert yml configuration.

If the table does not exist in Snowflake, the table will be created with the custom columns when the meshlet is run. If the table already exists, the table will be altered to add the custom columns into the table.

Go to top of page

Compare tables from ServiceNow to Snowflake

Table Compare allows you to compare tables from one ServiceNow instance to Snowflake. This is useful because you can see each table's record count, as well as a list of record discrepancies by **sys_id** between the two tables you're comparing—in other words, you can see which records exist in one table but not the other. To get started with comparing your ServiceNow tables to Snowflake, see Table Compare: ServiceNow to database table compare.

NOTE: This requires version Helium and above Perspectium Core update set.

Go to top of page

Load Schema Files

This feature allows you to load your ServiceNow table schemas to the meshlet when a connection cannot be made to the originating ServiceNow instance. Schema files are used to create and update tables in Snowflake. Enabling this feature will force the meshlet to read schemas via files exclusively from the local file storage and disable all calls to ServiceNow. Any new tables shared will require table schemas to be added to the local file storage.

NOTE: By default, this option will load all schemas stored in local file storage when the meshlet starts up to create tables in the Snowflake database immediately so as to improve performance when the meshlet is processing records. If you only want tables to be created when a record is received for a table, see the loadSchemasOnStart configuration.

1. Export the table schemas in your ServiceNow instance. See Download Table Schemas.

2. Unzip the folder that was created from the previous step. Then, put the folder in the *static/config* directory.

   The final location for each table should look like *static/config/schemas/incident.xml*.

3. In the application-dev.yml, set the following configurations:

```
perspectium:
        auth:
                useLocalSchema: true
```

   NOTE: By default or if the configuration is not specified, **useLocalSchema** is set to false.

Go to top of page

Azure External Storage

To start using Microsoft Azure as an external stage for bulk loading, follow these steps:

1. In the application-dev.yml, set the following configurations:

Directive	Description

url	Azure storage URL.
	`perspectium:`
	`  azure:`
	`  url: azure://pspsnowflaketest.blob.core.windows.net/snowflakedev`
	Where **pspsnowflaketest** is the name of the Azure storage account and ***snowflakedev*** is the name of the Azure container.

sasToken	Shared Access Signatures (SAS) connection string for Azure Storage. See SAS connection string.
	
	To access the connection string, go to Storage Account > Account Name > Shared Access Signature. Then enter the required fields, and generate SAS and connection String.
	
	Allowed Resource Types:
	○ Container
	○ Object
	
	Allowed Permissions:
	○ Read
	○ Write
	○ Delete
	○ List
	○ Add
	○ Create
	
	`perspectium:`
	`azure:`
	`sasToken: ?sv=2020-08-04&ss…..ejl%2BTE%3D`

connectionString	Connection URL for your Azure. To access the URL, go to Azure Portal > Storage Account > Access Keys > Show Keys > Connection String.
	
	`perspectium:`
	`azure:`
	`connectionString: DefaultEndpointsProtocol=.....EndpointSuffix=core.windows.net`

destinationContainer	Azure container you want to share your data to.
	
	`perspectium:`
	`azure:`
	`destinationContainer: snowflakedev`

deleteFiles	If you want to leave the temporary staging files in Azure. Setting this to **false** will not delete the staging files from Azure.
	
	`perspectium:`
	`fileSubscriber:`
	`deleteFiles: true`

Go to top of page

Soft Deletes

This feature allows you to add columns to a database table that gets updated when the meshlet receives a message indicating a record is to be deleted. The meshlet will update said columns appropriately and will not delete the record in Snowflake.

NOTE: At least one of the following table columns are required to achieve soft deletes, if neither column has been added to the configurations but **softDelete** is set to true, no delete will occur. This is to prevent deletion of records when there is an intention of using soft deletes.

1

In the application-dev.yml, set the following configurations:

```
perspectium:
        snowflake:
                softDelete: true
```

NOTE: By default or if the configuration is not specified, **softDelete** is set to false.

2

One option is to add a boolean field to mark a record as true or false as to whether it has been deleted.

In *static/config/databaseConfig.json*, add the following:

```
"delete": {
         "columnName":"is_delete"
 }
```

3

Another option is to add an IODateTime field to indicate the date and time the record was marked as deleted.

In *static/config/databaseConfig.json*, add the following:

```
"IODateTime": {
        "insert":["psp_per_insert_dt"],
        "update":["psp_per_update_dt"],
        "delete" :["psp_per_delete_dt"]
}
```

Go to top of page

Timestamp Columns and Timezones

Snowflake supports saving timestamp columns with different variations as described here.

For example, if you are using the **TIMESTAMP_NTZ** variation, records saved into a timestamp column will be saved with the timestamp value as is with no timezone offset possible. Thus when records are queried, Snowflake will always return the exact same timestamp value regardless of the timezone you specify in your session using the **TIMEZONE** parameter.

That is, if you save a value as **2021-01-01 00:14:30** into a column defined as **TIMESTAMP_NTZ(9)**, Snowflake will always return the value as **2021-01-01 00:14:30** regardless of the timezone your current session is in.

To mimic how data is saved in ServiceNow, by default the meshlet uses the **TIMEZONE** session parameter in its JDBC connection string when connecting to Snowflake, specifying that timestamp values the meshlet saves will be in UTC (**TIMEZONE=UTC**). In addition to using this parameter, **glide_date_time** fields from ServiceNow (which is the default field type for datetime fields in ServiceNow) are mapped to the **TIMESTAMP_LTZ(9)** column type in the meshlet's **databaseConfig.json** configuration file.

This combination ensures that ServiceNow timestamp fields the meshlet pushes into Snowflake are saved in UTC time and then allows for querying of records to be returned in local timezone based on the TIMEZONE parameter a user sets in their session.

That is, if you save a value as **2021-01-01 00:14:30** and specified that we were saving it in UTC time (**TIMEZONE=UTC**) into a column defined as **TIMESTAMP_LTZ(9)**, Snowflake will return the value as **2021-01-01 00:06:30 -0800** if your current session is in the America/Los_Angeles (Pacific) timezone.

Change meshlet to not save in UTC

If you prefer the meshlet to not save records in UTC time, set the following configuration in the application-dev.yml meshlet configuration file:

NOTE: This is an advanced configuration and it is recommended you review how Snowflake handles timestamp variations before making these changes.

```
perspectium:
        snowflake:
                useUTCTimeZone: false
```

By default or if the configuration is not specified, **useUTCTimeZone** is set to **true** so records are saved in UTC time.

When setting this to false, the meshlet will save in the session timezone as defined in the session/account for the Snowflake account credentials entered in application-dev.yml for connecting to Snowflake.

After setting this configuration to false, you can then use the **connectionUrl** configuration in application-dev.yml to use a different timezone if desired i.e.

```
perspectium:
        snowflake:
                connectionUrl: jdbc:snowflake://<account_identifier>.snowflakecomputing.com/?TIMEZONE=America
/Los_Angeles
```

In this case, records will be saved into the database with the America/Los_Angeles (Pacific) timezone.

Change timestamp column format

You can modify the **glide_date_time** configuration in the **databaseConfig.json** file if you want timestamp fields to be saved in a different **TIMESTAMP** variation and precision.

NOTE: This is an advanced configuration and it is recommended you review how Snowflake handles timestamp variations before making these changes.

For example, if you wanted to change all ServiceNow datetime fields to be saved without timezone considerations with a precision of 0 (only seconds), you can update the **glide_date_time** configuration in **databaseConfig.json** as follows:

```
"types":{
      "glide_date_time":"TIMESTAMP_NTZ(0)",
 }
```

Go to top of page

Error Logging

Starting with the **Helium 6.1.4 release**, the meshlet will log details on when it fails with a record in the batch, listing the error message as returned by Snowflake. For example:

```
2022-05-20 20:25:36,197 ERROR [task-2] com.perspectium.meshlet.common.DataSyncReceiptService: Error
processing record: {"sys_id":"0c611d5e1b27011003b30f26624bcbba","table":"ticket","message":"User character
length limit (5000) exceeded by string 'Issue with my email, can you please take a look'"}
2022-05-20 20:25:39,225 INFO [scheduling-1] com.perspectium.meshlet.listener.SnowflakeListenerService:
Processed 5 record(s) to table: ticket with 1 record(s) having errors
```

Go to top of page