

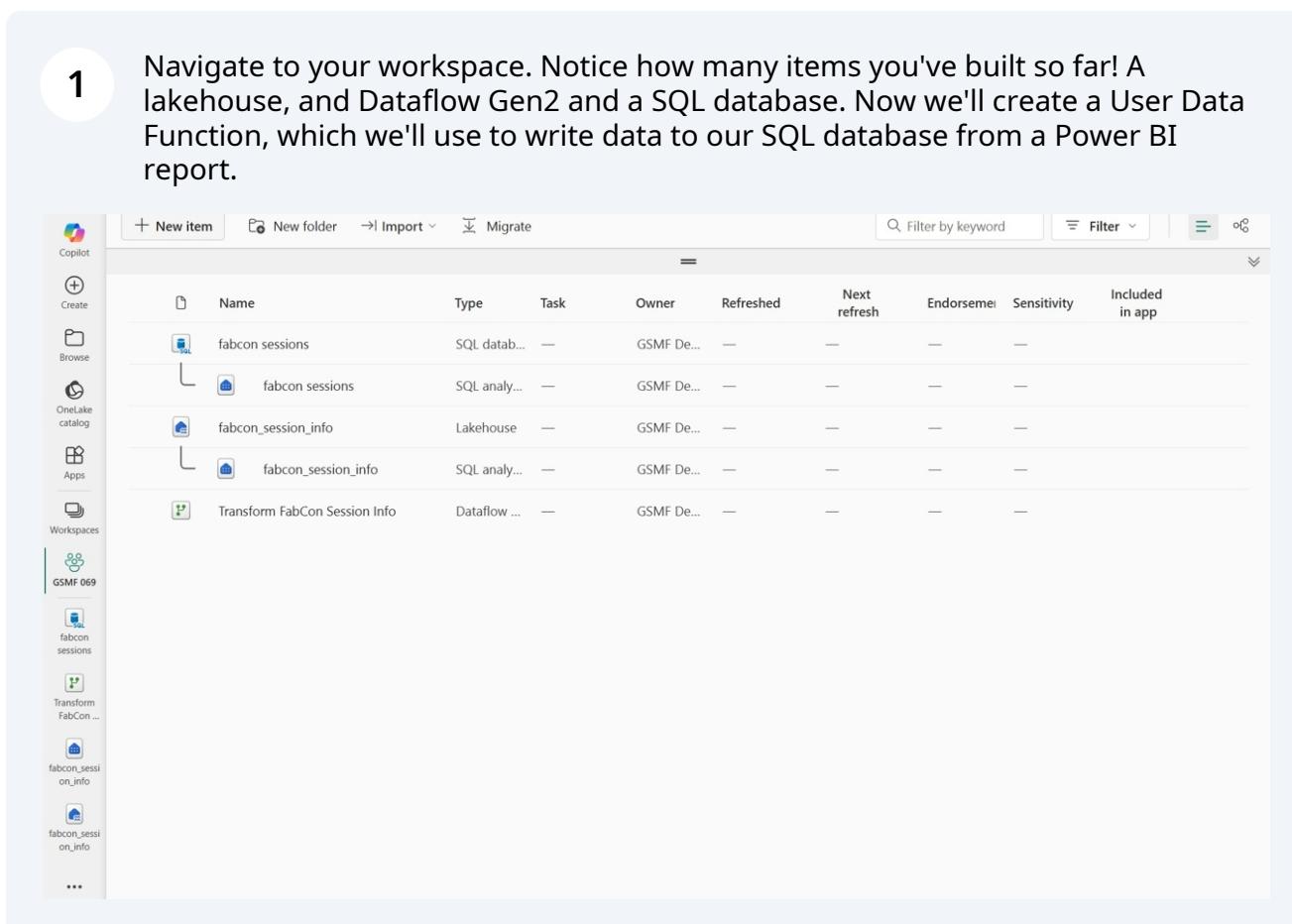
3b. Create and Test a Fabric User Data Functions

Training. 

Create a new function

1

Navigate to your workspace. Notice how many items you've built so far! A lakehouse, and Dataflow Gen2 and a SQL database. Now we'll create a User Data Function, which we'll use to write data to our SQL database from a Power BI report.



The screenshot shows the Microsoft Fabric workspace interface. On the left, there's a sidebar with icons for Copilot, Create, Browse, OneLake catalog, Apps, Workspaces, and a workspace named 'GSMF-069'. The main area displays a table of items:

Name	Type	Task	Owner	Refreshed	Next refresh	Endorsement	Sensitivity	Included in app
fabcon sessions	SQL database	—	GSMF De...	—	—	—	—	—
fabcon sessions	SQL analysis	—	GSMF De...	—	—	—	—	—
fabcon_session_info	Lakehouse	—	GSMF De...	—	—	—	—	—
fabcon_session_info	SQL analysis	—	GSMF De...	—	—	—	—	—
Transform FabCon Session Info	Dataflow ...	—	GSMF De...	—	—	—	—	—

2 Click "New item"

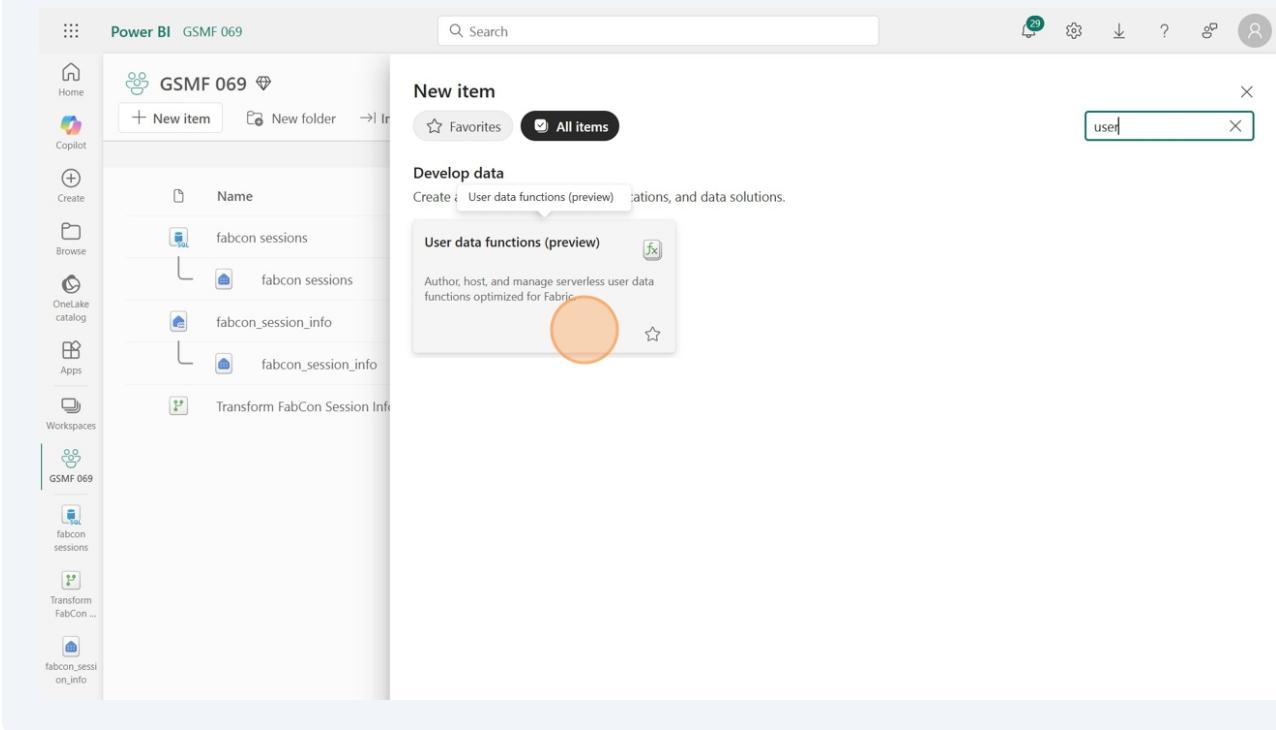
The screenshot shows the Power BI workspace interface for a workspace named 'GSMF 069'. On the left, there's a sidebar with various navigation options like Home, Copilot, Create, Browse, OneLake catalog, Apps, and Workspaces. The main area displays a table with several items, each with a small icon, name, type, task, owner, refresh status, next refresh, endorser, sensitivity, and inclusion in app. A prominent orange circle highlights the '+ New item' button at the top left of the table header. The table contains the following data:

	Name	Type	Task	Owner	Refreshed	Next refresh	Endorser	Sensitivity	Included in app
	fabcon sessions	SQL datab...	—	GSMF De...	—	—	—	—	—
	fabcon sessions	SQL analy...	—	GSMF De...	—	—	—	—	—
	fabcon_session_info	Lakehouse	—	GSMF De...	—	—	—	—	—
	fabcon_session_info	SQL analy...	—	GSMF De...	—	—	—	—	—
	Transform FabCon Session Info	Dataflow ...	—	GSMF De...	—	—	—	—	—

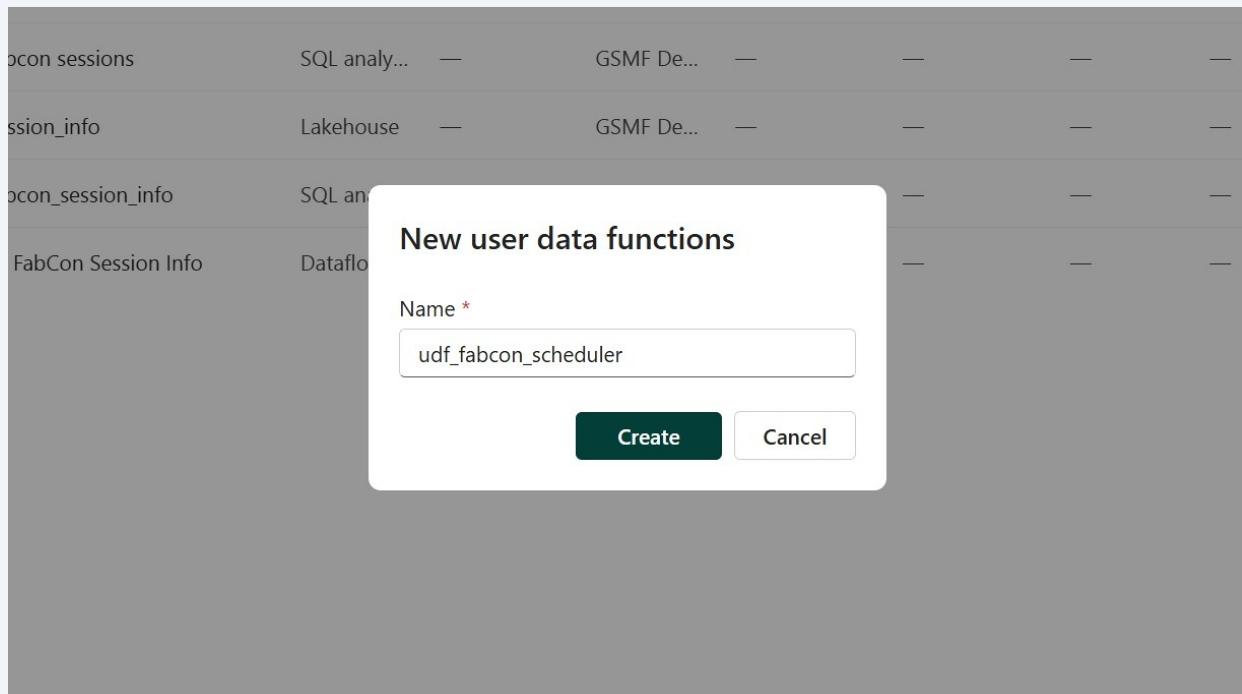
3 Click the "Filter by item type" field and type "user" to easily locate the user data function.

The screenshot shows the 'New item' dialog box within the Power BI workspace. The dialog has tabs for 'Favorites' (disabled) and 'All items' (selected). A search bar labeled 'Filter by item type' is highlighted with an orange circle. Below the tabs, there are sections for 'Visualize data', 'Get data', and 'Scorecard'. Under 'Visualize data', there are cards for 'Dashboard', 'Exploration (preview)', 'Org app (preview)', 'Paginated Report (preview)', 'Real-Time Dashboard', and 'Report'. Each card provides a brief description and a star icon for favoriting. The 'Get data' section is partially visible at the bottom.

- 4 Click the "User data functions (preview)" tile.

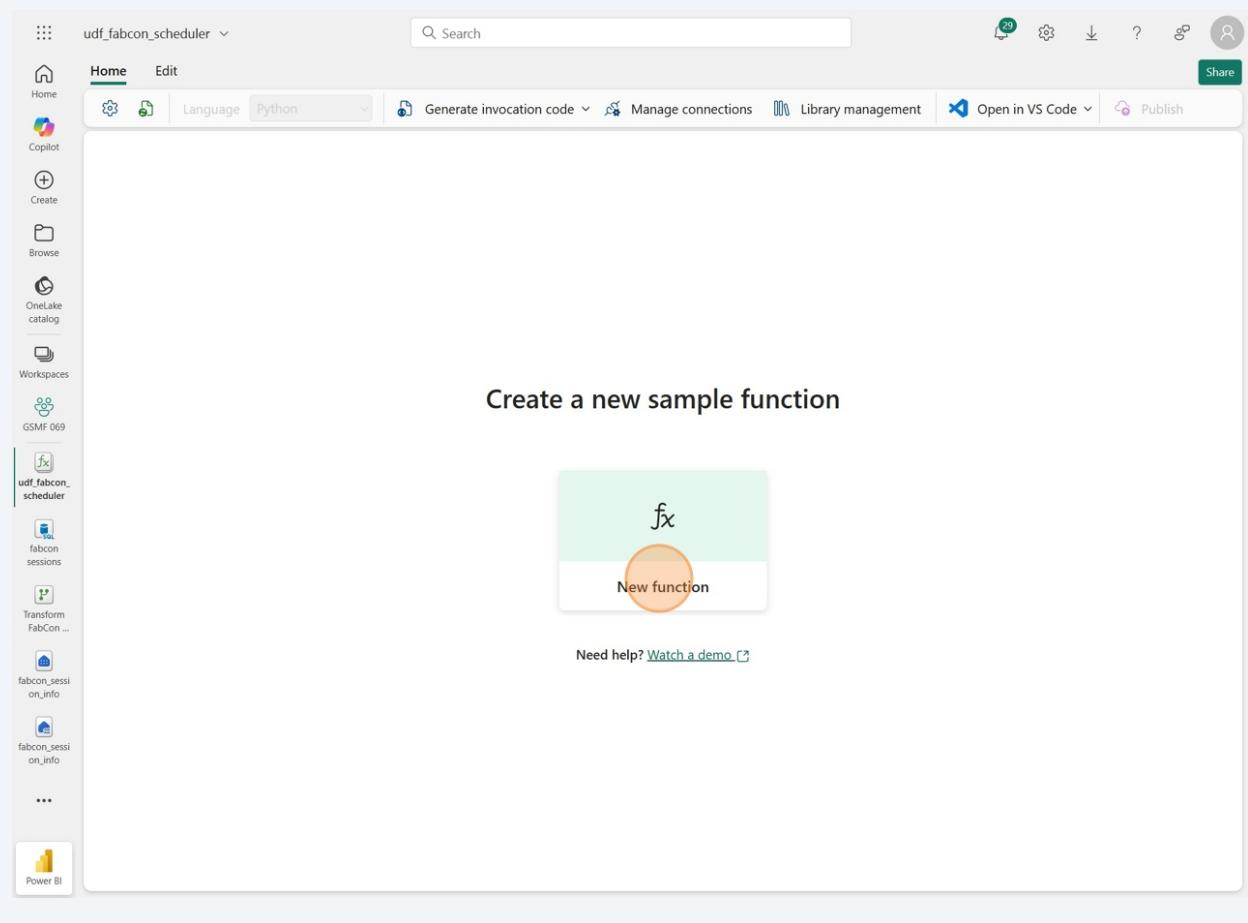


- 5 Give the function a name like "udf_fabcon_scheduler", for example, and click the "Create" button.



6

Now you're in the user data function editor. Click the big "New function" button in the middle of the canvas.



7

A sample function is automatically created. It doesn't do much other than return a message based on an input, but it gives you an idea of what the code should look like. We'll test this sample function to see what happens.

```

import datetime
import fabric.functions as fn
import logging

udf = fn.UserDataFunctions()

@udf.function()
def hello_fabric(name: str) -> str:
    logging.info('Python UDF trigger function processed a request.')
    return f"Welcome to Fabric Functions, {name}, at {datetime.datetime.now()}"

```

Test the sample function

8

Hover over the "hello_fabric" function and click the little test beaker to test the function.

```

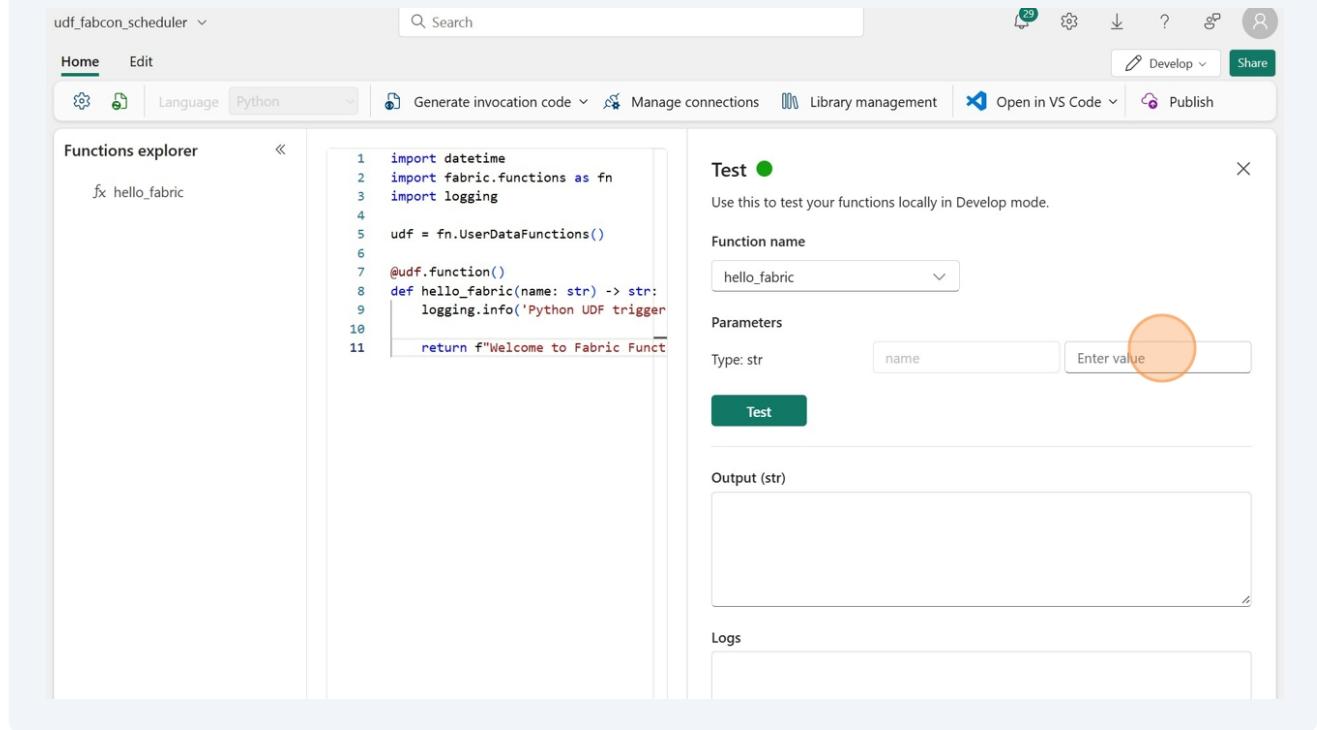
import datetime
import fabric.functions as fn
import logging

udf = fn.UserDataFunctions()

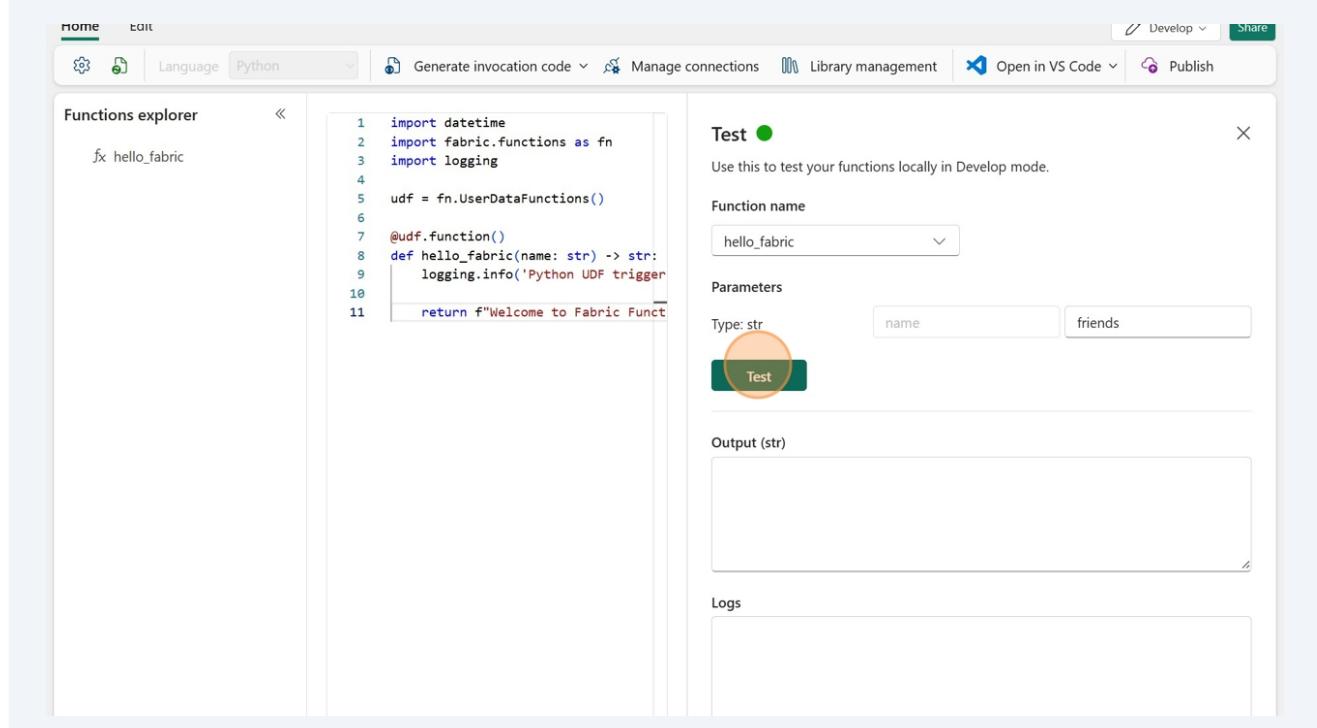
@udf.function()
def hello_fabric(name: str) -> str:
    logging.info('Python UDF trigger function processed a request.')
    return f"Welcome to Fabric Functions, {name}, at {datetime.datetime.now()}"

```

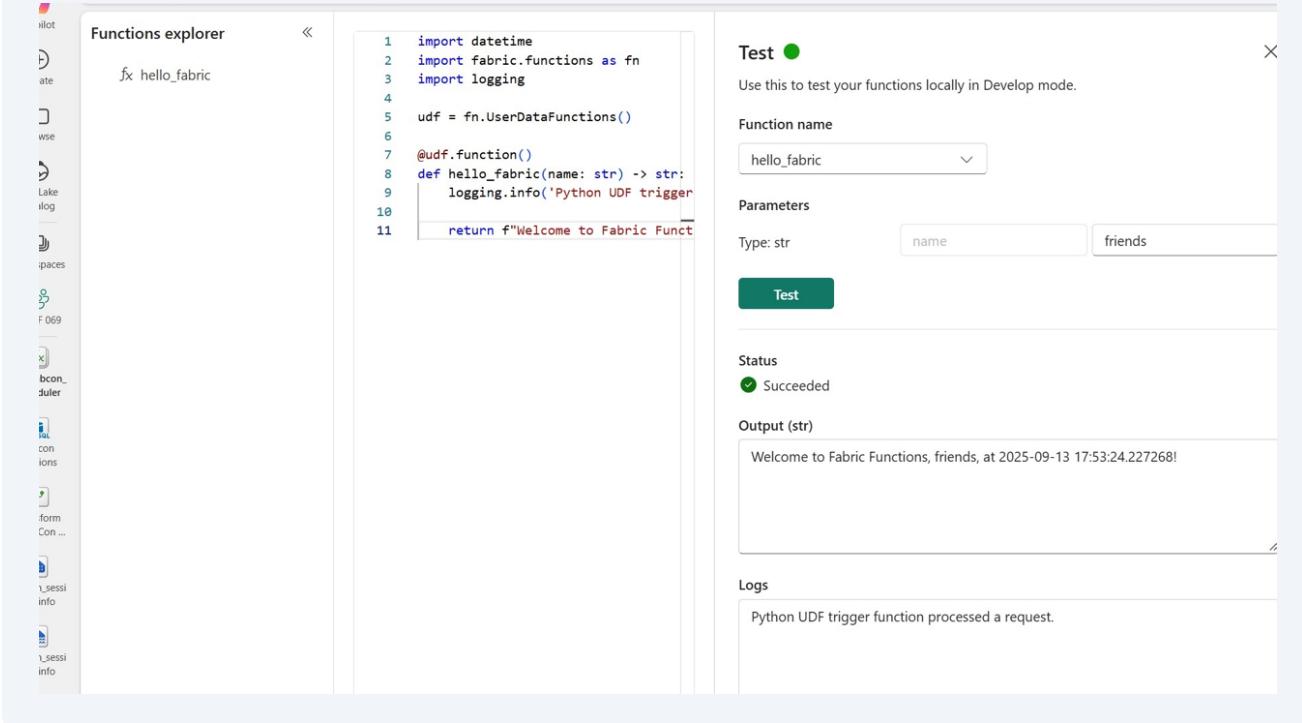
9 Click the "Enter value" field and enter you name.



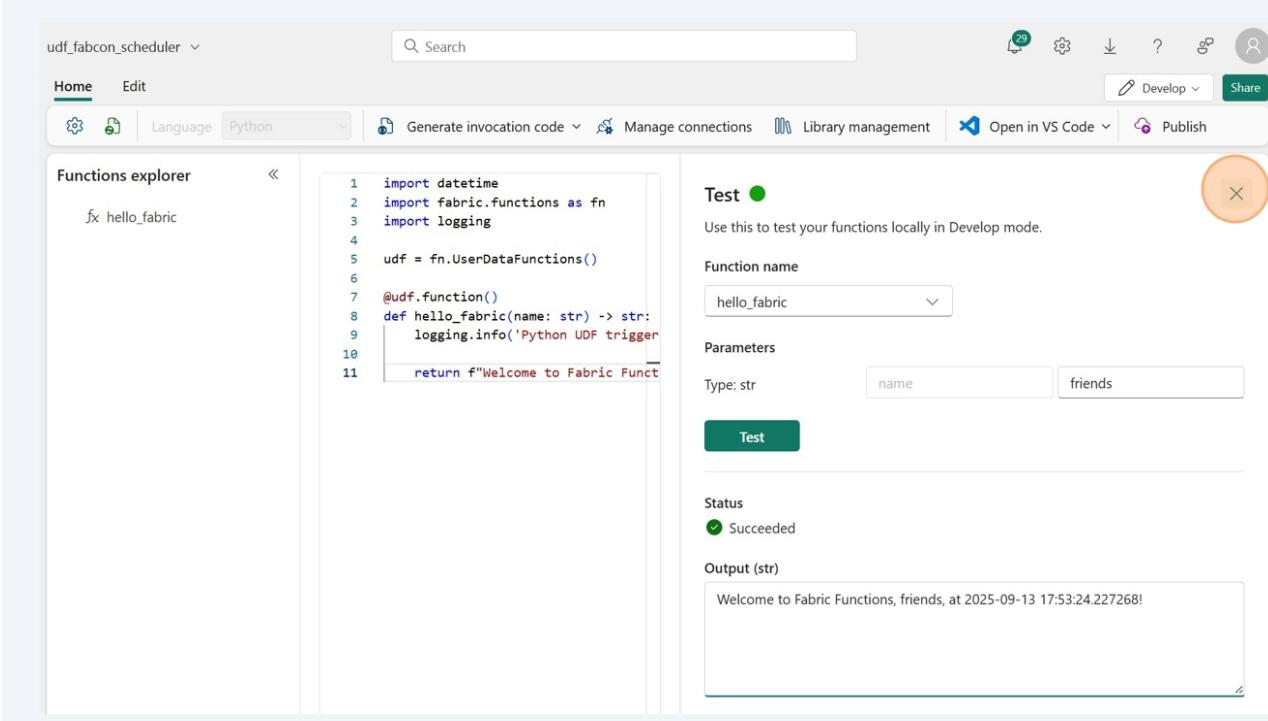
10 Click the "Test" button. (make sure it gets clicked and doesn't just move your cursor)



11 After a few seconds you should see the output and any logging information. It may take a few seconds for it to kick off, so be patient.



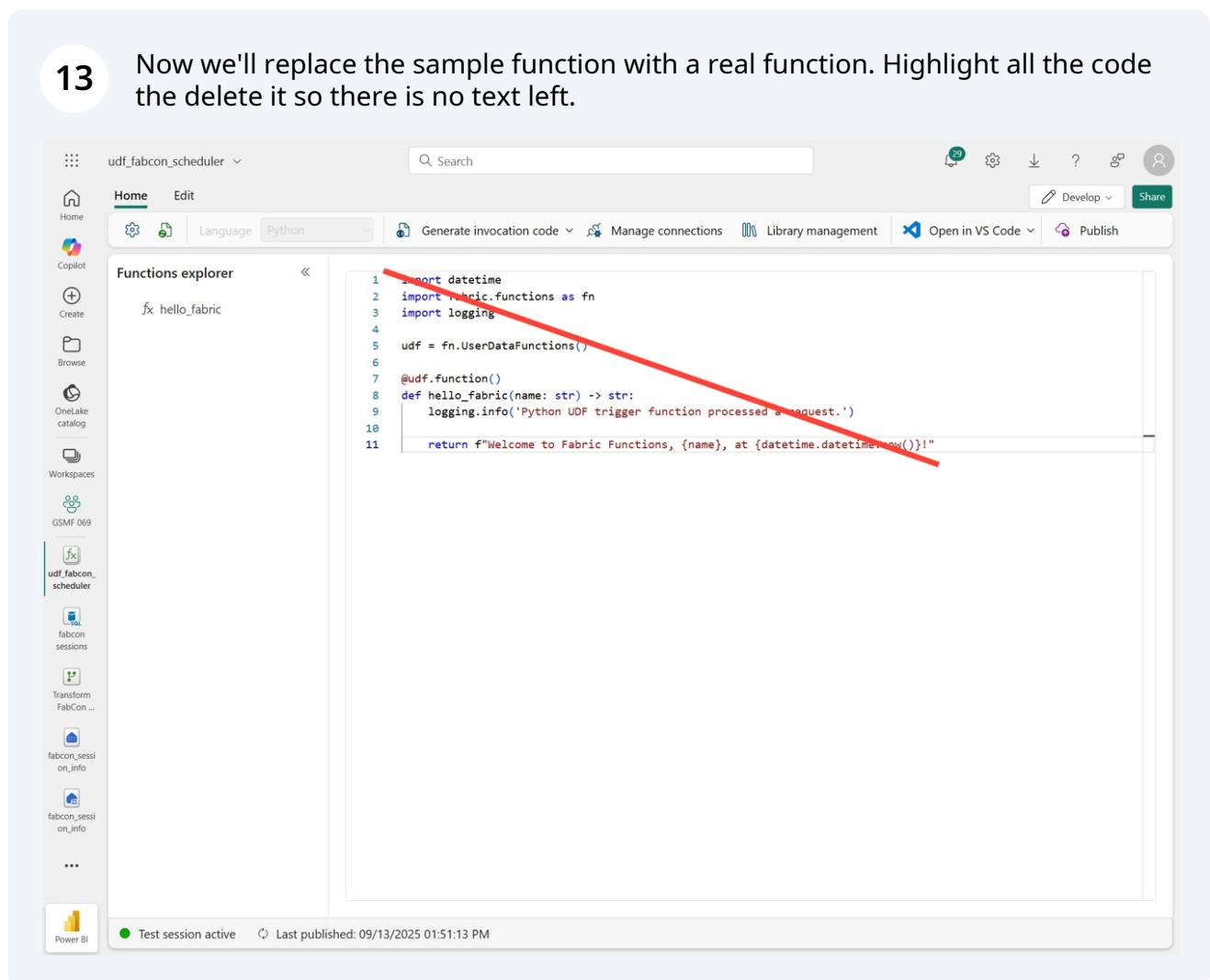
12 Close the test pane by clicking the "X" in the top right.



Enter the code for the functions

13

Now we'll replace the sample function with a real function. Highlight all the code then delete it so there is no text left.



The screenshot shows the Fabric Functions interface. On the left is a sidebar with various project and session icons. The main area is titled 'Functions explorer' and contains a file named 'fx hello_fabric'. The code editor displays the following Python code:

```
1 import datetime
2 import fabric.functions as fn
3 import logging
4
5 udf = fn.UserDataFunctions()
6
7 @udf.function()
8 def hello_fabric(name: str) -> str:
9     logging.info('Python UDF trigger function processed a request.')
10
11     return f"Welcome to Fabric Functions, {name}, at {datetime.datetime.now()}!"
```

A large red diagonal line is drawn across the entire code block, indicating that the user should highlight and delete this text.

14

Open the UDF.txt file from the resources section and copy all the code. Paste it into the UDF editor.

15

Your function should now look like this. In fact, we have two functions (which you can see in the Functions explorer on the left). One for adding a session to your agenda, and one for deleting it from your agenda. Take a look at the code to get an idea of how it works.

The screenshot shows the Azure Functions developer portal interface. On the left, the 'Functions explorer' sidebar lists several functions: 'udf.fabcon.scheduler', 'fabcon_sessions', 'Transform FabCon ...', 'fabcon_session_info', 'fabcon_session_on_info', and '...'. The main area is a code editor with Python syntax highlighting. The code is as follows:

```
48 cursor.execute(merge_query, merge_params)
49 logging.info("Attendee was added/modified")
50
51 # Commit the transaction
52 connection.commit()
53
54 # Close the connection
55 cursor.close()
56 connection.close()
57 return "Attendee table was created (if necessary) and data was added to this table"
58
59 @udf.connection(argName="sqlDB",alias="fabconinfo")
60 @udf.function()
61 def delete_from_agenda(sqlDB: fn.FabricSqlConnection, sessionid: int, Attendee: str) -> str:
62
63     data = (sessionid, Attendee)
64
65     # Establish a connection to the SQL database
66     connection = sqlDB.connect()
67     cursor = connection.cursor()
68
69     logging.info("Removing attendee")
70     # Insert data into the table
71     insert_query = "DELETE FROM dbo.attendees WHERE session_id = ? and Attendee = ?;""
72     cursor.execute(insert_query, data)
73     logging.info("Attendee was removed")
74
75     # Commit the transaction
76     connection.commit()
77
78     # Close the connection
79     cursor.close()
80     connection.close()
81
82     return "Attendee was removed"
83
```

16

Notice there are two places where the text has a squiggly red line. You can see in the scrollbar on the right side that there are two little red squares indicating errors.

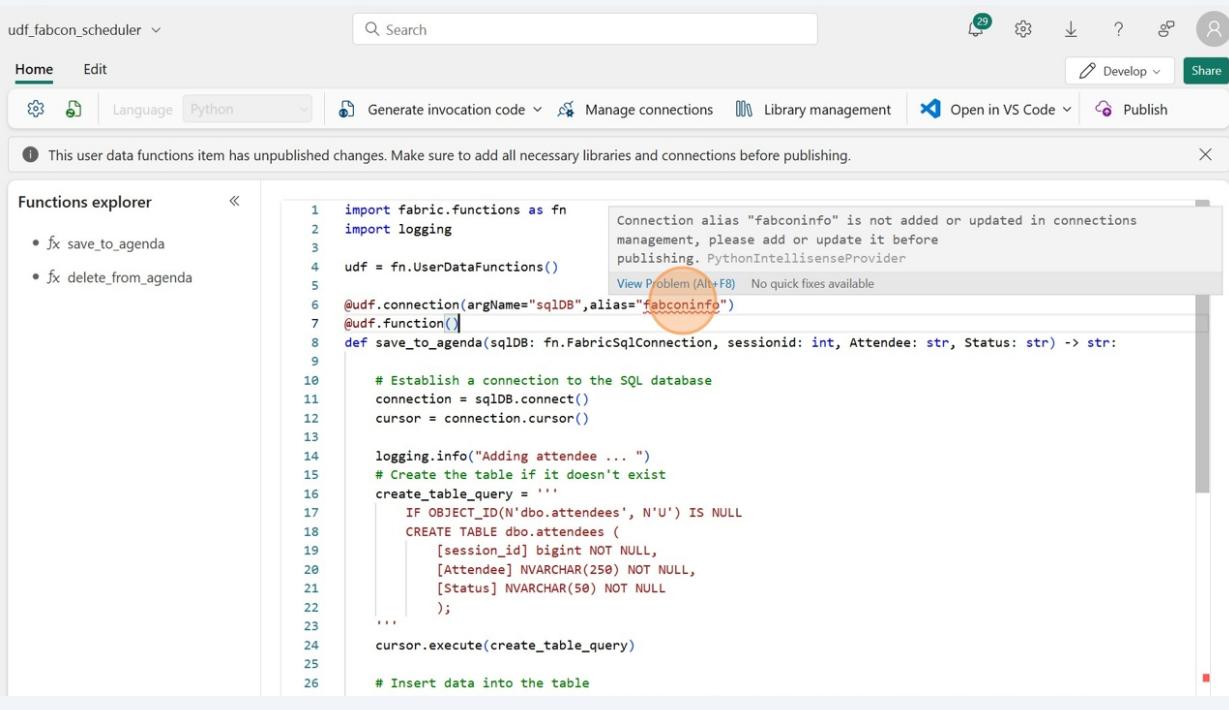
The screenshot shows the Power BI Editor interface with the 'Edit' tab selected. On the left, the 'Functions explorer' pane lists two functions: `fx save_to_agenda` and `fx delete_from_agenda`. The main workspace displays a Python script:

```
1 import fabric.functions as fn
2 import logging
3
4 udf = fn.UserDataFunctions()
5
6 #udf.connection(argName="sqlDB",alias="fabconinfo")
7 #udf.function()
8 def save_to_agenda(sqlDB: fn.FabricSqlConnection, sessionid: int, Attendee: str, Status: str) -> str:
9
10     # Establish a connection to the SQL database
11     connection = sqlDB.connect()
12     cursor = connection.cursor()
13
14     logging.info("Adding attendee ... ")
15     # Create the table if it doesn't exist
16     create_table_query = """
17         IF OBJECT_ID('N'dbo.attendees', N'U') IS NULL
18             CREATE TABLE dbo.attendees (
19                 [session_id] bigint NOT NULL,
20                 [Attendee] NVARCHAR(250) NOT NULL,
21                 [Status] NVARCHAR(50) NOT NULL
22             );
23
24     cursor.execute(create_table_query)
25
26     # Insert data into the table
27     # insert_query = "INSERT INTO dbo.attendees (session_id, Attendee, Status) VALUES (?, ?, ?);"
28     merge_query = """
29         MERGE dbo.attendees AS target
30             USING (SELECT ? AS session_id, ? AS Attendee) AS src
31             ON target.session_id = src.session_id AND target.Attendee = src.Attendee
32             WHEN MATCHED THEN
33                 UPDATE SET Status = ?
34             WHEN NOT MATCHED THEN
35                 INSERT (session_id, Attendee, Status)
36                     VALUES (?, ?, ?);
```

Two specific lines of code have red squiggly underlines: `#udf.connection(argName="sqlDB",alias="fabconinfo")` and `cursor.execute(create_table_query)`. The scrollbar on the right side of the code editor has two small red squares indicating errors at these positions.

17

If you click the error, you'll see a message telling you that we are referring to a connection that doesn't exist. We need to create a connection to our SQL database for this function to use as the connection.



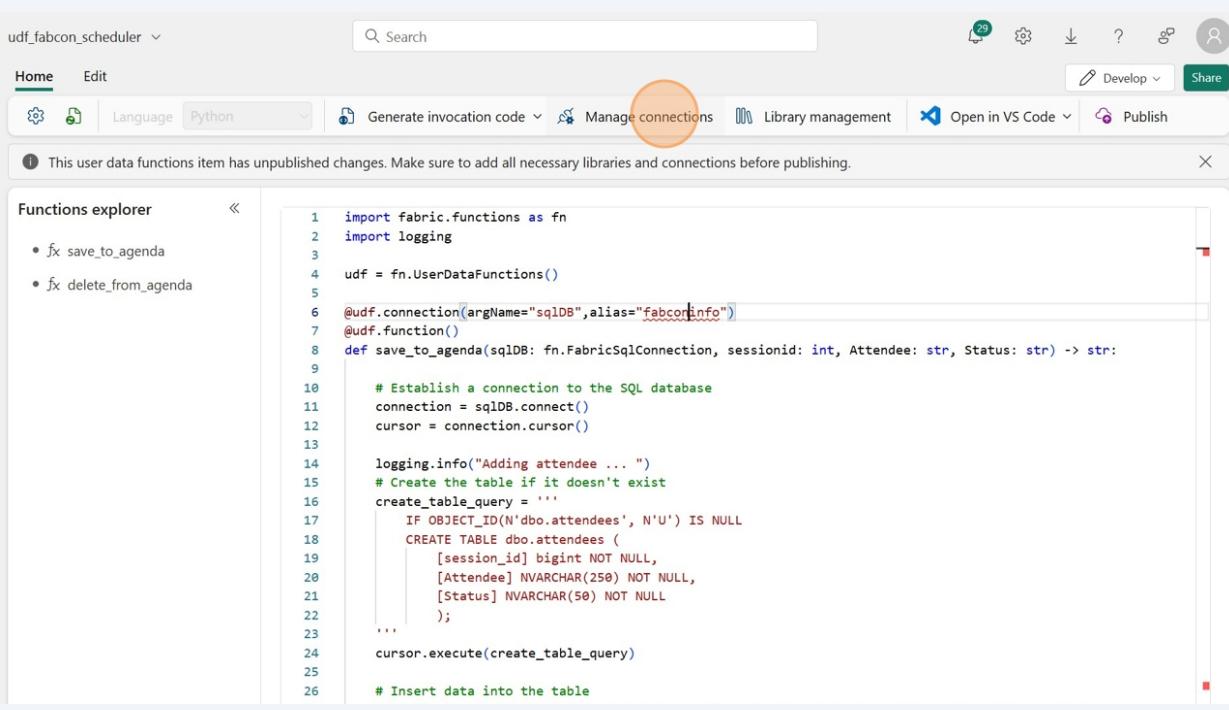
```
import fabric.functions as fn
import logging
udf = fn.UserDataFunctions()
@udf.connection(argName="sqlDB", alias="fabconinfo")
@udf.function()
def save_to_agenda(sqlDB: fn.FabricSqlConnection, sessionid: int, Attendee: str, Status: str) -> str:
    # Establish a connection to the SQL database
    connection = sqlDB.connect()
    cursor = connection.cursor()
    logging.info("Adding attendee ... ")
    # Create the table if it doesn't exist
    create_table_query = """
        IF OBJECT_ID('dbo.attendees', N'U') IS NULL
        CREATE TABLE dbo.attendees (
            [session_id] bigint NOT NULL,
            [Attendee] NVARCHAR(250) NOT NULL,
            [Status] NVARCHAR(50) NOT NULL
        );
    """
    cursor.execute(create_table_query)
    # Insert data into the table
```

This screenshot shows the Azure Functions portal interface. The code editor contains Python code for a User Data Function (UDF) named `save_to_agenda`. The code connects to a SQL database using a connection named `fabconinfo`. A tooltip is displayed over the line `@udf.connection(argName="sqlDB", alias="fabconinfo")`, stating: "Connection alias 'fabconinfo' is not added or updated in connections management, please add or update it before publishing. PythonIntellisenseProvider". The tooltip also includes a "View Problem (Alt+F8)" link and a note that "No quick fixes available".

Create a connection to the SQL database

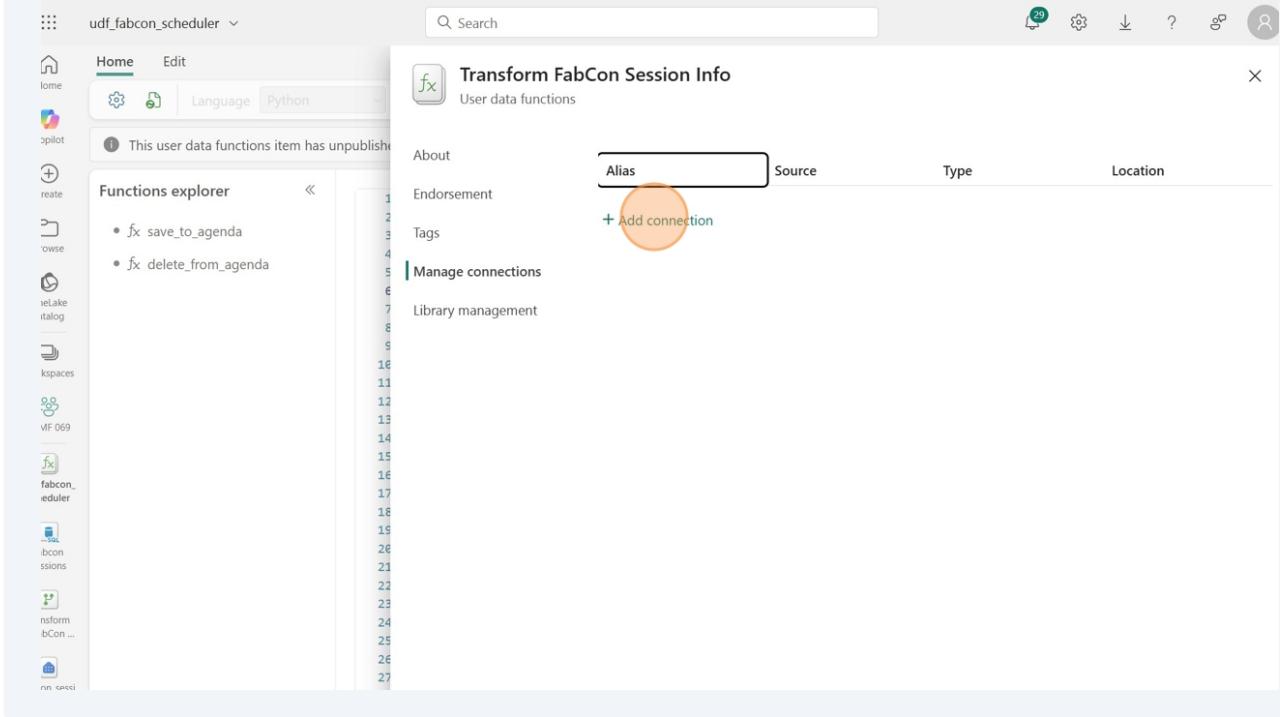
18

Click "Manage connections"



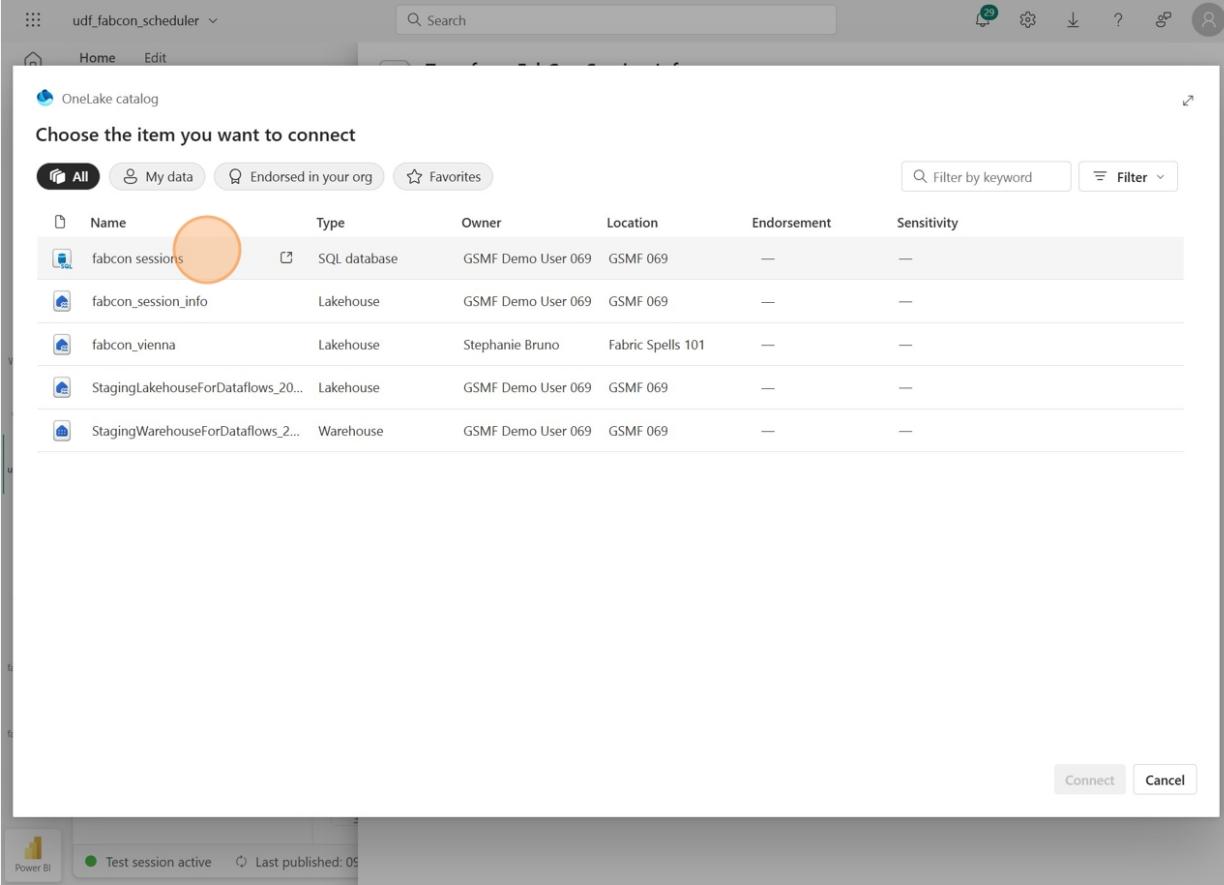
This screenshot shows the same Azure Functions portal interface as the previous one, but with the "Manage connections" button highlighted by a large orange circle. The code editor and tooltip are identical to the previous screenshot, indicating that the connection alias `fabconinfo` is still not defined.

19 Click "Add connection"



20

Click on the SQL database you created in the previous lab. This is the one we want to connect to.

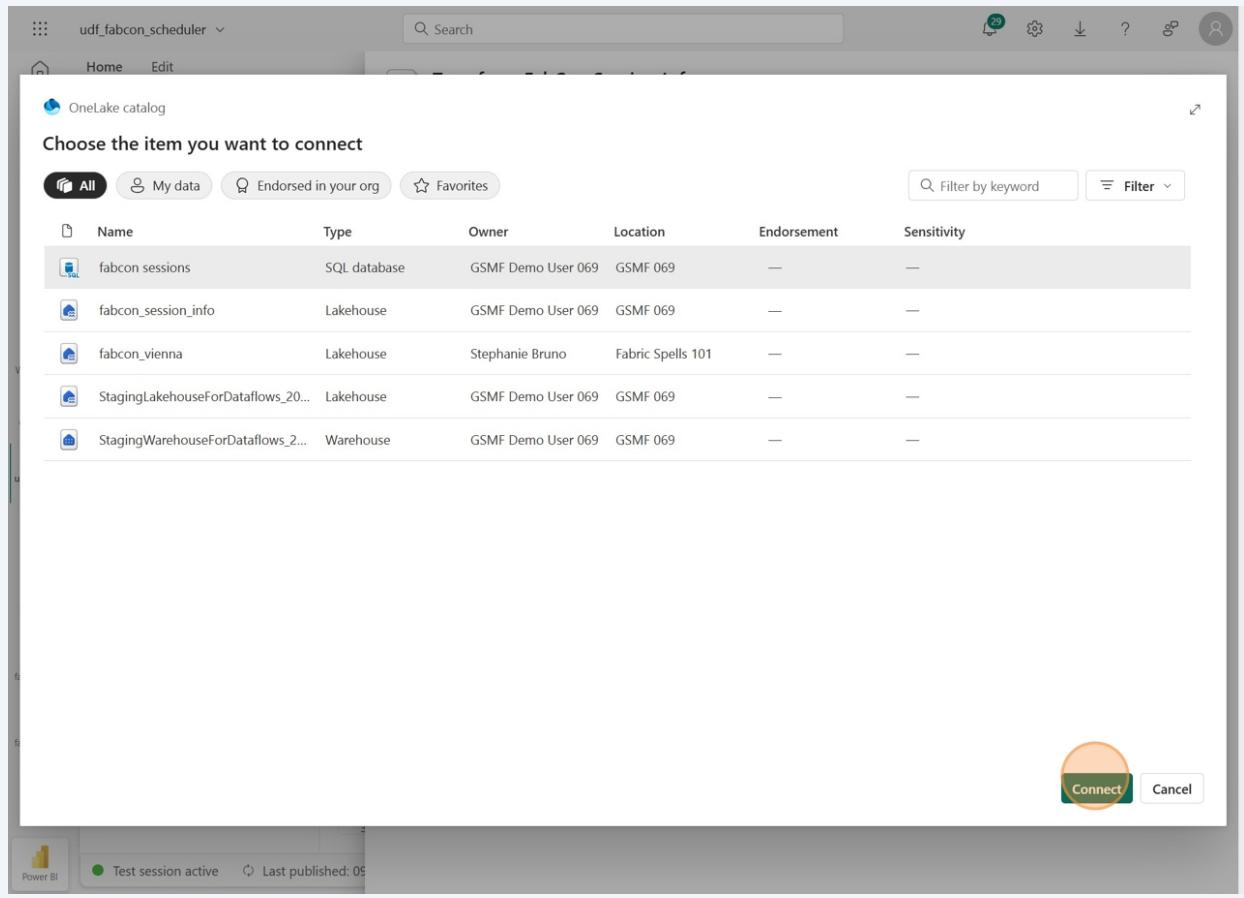


The screenshot shows the OneLake catalog interface. At the top, there's a search bar and some navigation icons. Below that, a header bar with 'Home' and 'Edit' buttons. The main area is titled 'Choose the item you want to connect'. It features a table with columns: Name, Type, Owner, Location, Endorsement, and Sensitivity. There are filters for 'All', 'My data', 'Endorsed in your org', and 'Favorites', along with a search bar and a 'Filter' button. The table lists five items:

Name	Type	Owner	Location	Endorsement	Sensitivity
fabcon sessions	SQL database	GSMF Demo User 069	GSMF 069	—	—
fabcon_session_info	Lakehouse	GSMF Demo User 069	GSMF 069	—	—
fabcon_vienna	Lakehouse	Stephanie Bruno	Fabric Spells 101	—	—
StagingLakehouseForDataflows_20...	Lakehouse	GSMF Demo User 069	GSMF 069	—	—
StagingWarehouseForDataflows_2...	Warehouse	GSMF Demo User 069	GSMF 069	—	—

At the bottom right of the modal are 'Connect' and 'Cancel' buttons. The footer of the interface shows 'Power BI' and status information: 'Test session active' and 'Last published: 09'.

21 Click "Connect"



22

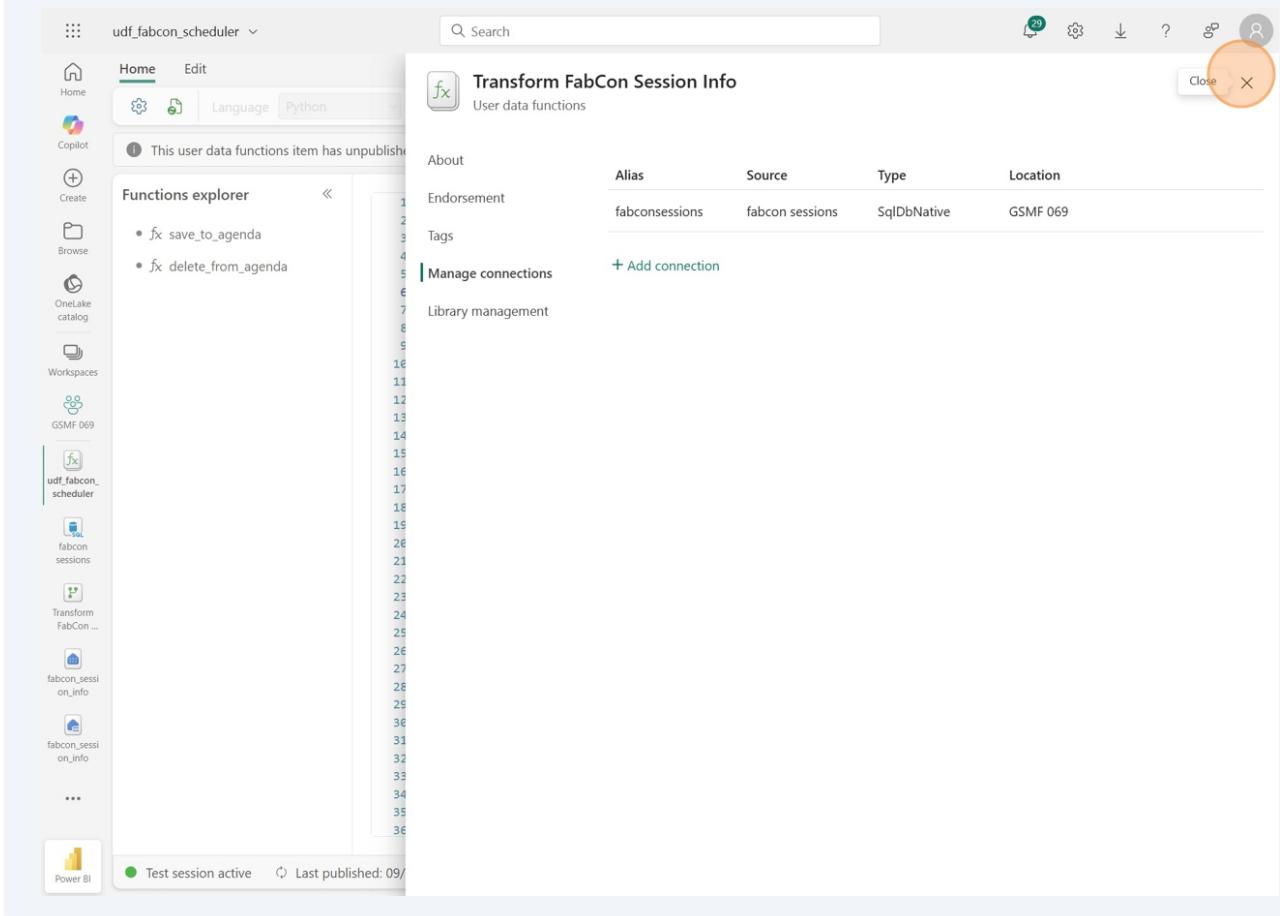
You should now see one connection listed. The source is your SQL database and the type is SqlDbNative. The location is your workspace. It automatically created an alias for the connection based on your SQL database's name. You can either edit that alias or accept it as-is. Either way, the alias is what we need the code to refer to in the two lines using the connection. Copy the alias name so you can update the code with this value.

The screenshot shows the Data Studio interface with the following details:

- Left sidebar:** Shows various projects and datasets, including "udf_fabcon_scheduler", "elake", "AF 069", "fabcon_sessions", "transform_bCon ...", and "sn_secc".
- Top navigation:** Home, Edit, Language (Python), and a search bar.
- Central area:** A card titled "Transform FabCon Session Info" with the sub-section "User data functions".
- Manage connections table:** A table with columns: Source, Type, and Location. It contains one row:

Source	Type	Location
fabconsessions	SqlDbNative	GSMF 069
- Left sidebar (continued):** Functions explorer showing "fx save_to_agenda" and "fx delete_from_agenda".
- Bottom sidebar:** A vertical list of numbers from 1 to 27.

23 Click the "X" to close the connections pane.



24

Double-click "fabconinfo" on line 6 of the code and replace it with whatever your connection's alias name is.

```
import fabric.functions as fn
import logging
udf = fn.UserDataFunctions()
@udf.connection(argName="sqlDB", alias="fabconinfo")
@udf.function()
def save_to_agenda(sqlDB: fn.FabricSqlConnection, sessionid: int, Attendee: str, Status: str) -> str:
    # Establish a connection to the SQL database
    connection = sqlDB.connect()
    cursor = connection.cursor()
    logging.info("Adding attendee ... ")
    # Create the table if it doesn't exist
    create_table_query = """
        IF OBJECT_ID('N'dbo.attendees', N'U') IS NULL
        CREATE TABLE dbo.attendees (
            [session_id] bigint NOT NULL,
            [Attendee] NVARCHAR(250) NOT NULL,
            [Status] NVARCHAR(50) NOT NULL
        );
    """
    cursor.execute(create_table_query)
    # Insert data into the table
```

25

Do the same on line 59. After these two replacements, you shouldn't see the little red marks on the right-side scroll bar anymore.

```
def delete_from_agenda(sqlDB: fn.FabricSqlConnection, sessionid: int, Attendee: str) -> str:
    data = (sessionid, Attendee)
    # Establish a connection to the SQL database
    connection = sqlDB.connect()
    cursor = connection.cursor()
```

Test a function

26

Now we're ready to test! Click the "Test" icon next to the "save_to_agenda" function name.

This user data functions item has unpublished changes. Make sure to add all necessary libraries and connections before publishing.

```
import fabric.functions as fn
import logging

udf = fn.UserDataFunctions()

@udf.connection(argName="sqlDB", alias="fabconsessions")
@udf.function()
def save_to_agenda(sqlDB: fn.FabricSqlConnection, sessionid: int, Attendee: str, Status: str) -> str:
    # Establish a connection to the SQL database
    connection = sqlDB.connect()
    cursor = connection.cursor()

    logging.info("Adding attendee ... ")
    # Create the table if it doesn't exist
    create_table_query = """
        IF OBJECT_ID(N'dbo.attendees', N'U') IS NULL
        CREATE TABLE dbo.attendees (
            [session_id] bigint NOT NULL,
            [Attendee] NVARCHAR(250) NOT NULL,
            [Status] NVARCHAR(50) NOT NULL
        );
    """
    cursor.execute(create_table_query)

    # Insert data into the table
    # ...
```

27

The "save_to_agenda" takes three parameters: sessionid, Attendee, and Status. For testing purposes, it doesn't really matter what you enter, but sessionid must be an integer.

Enter values for each of the three parameters and click the "Test" button.

The screenshot shows the Azure Data Studio interface. On the left, the sidebar displays a workspace named 'GSMF-069' containing several items: 'udf_fabcon_scheduler', 'fabcon_sessions', 'Transform FabCon ...', 'fabcon_session_info', and 'fabcon_session_info'. The main area shows the 'Functions explorer' with two functions listed: 'fx save_to_agenda' and 'fx delete_from_agenda'. The code editor displays a Python script for the 'save_to_agenda' function:

```

1 import fabric.functions as fn
2 import logging
3
4 udf = fn.UserDataFunctions()
5
6 @udf.connection(argName="sqlDB", alias="udf.function()")
7 def save_to_agenda(sqlDB: fn.FabricS
8
9     # Establish a connection to the
10    connection = sqlDB.connect()
11    cursor = connection.cursor()
12
13    logging.info("Adding attendee ..")
14    # Create the table if it doesn't
15    create_table_query = """
16        IF OBJECT_ID(N'dbo.attendees'
17        CREATE TABLE dbo.attendees (
18            [session_id] bigint NOT
19            [Attendee] NVARCHAR(250)
20            [Status] NVARCHAR(50) NO
21            );
22
23    ...
24
25    cursor.execute(create_table_quer
26
27    # Insert data into the table
28    # insert_query = "INSERT INTO db
29    merge_query = """
30        MERGE dbo.attendees AS target
31        USING (SELECT ? AS session_i
32        ON target.session_id = src.s
33        WHEN MATCHED THEN
34            UPDATE SET Status = ?
35        WHEN NOT MATCHED THEN
36            INSERT (session_id, Atte
VALUES (?, ?, ?);

```

To the right, a 'Test' dialog box is open. It contains fields for 'Function name' (set to 'save_to_agenda'), 'Parameters' (with 'sessionid' set to 10, 'Attendee' set to 'me me me', and 'Status' set to 'Definitely'), and a 'Test' button which is highlighted with a yellow circle. Below the test area are sections for 'Output (str)' and 'Logs'.

28

While the function is running you can see the status is "In progress."

The screenshot shows the Azure Data Factory studio interface. On the left, the sidebar lists projects like 'udf_fabcon_scheduler', 'fabcon_sessions', 'Transform FabCom ...', 'fabcon_session_info', and 'fabcon_session_info'. The main area shows a Python script for a User Defined Function (UDF) named 'save_to_agenda'. The code uses the 'fabric' library to interact with a database. A test dialog is open, showing the function name 'save_to_agenda' and three parameters: 'sessionid' (value 10), 'Attendee' (value 'me me me'), and 'Status' (value 'Definitely'). The 'Status' field is highlighted with an orange box, and its value is 'In progress'. At the bottom of the test dialog, there are sections for 'Output (str)' and 'Logs'.

```

1 import fabric.functions as fn
2 import logging
3
4 udf = fn.UserDataFunctions()
5
6 @udf.connection(argName="sqlDB", alias="@udf.function()")
7 def save_to_agenda(sqlDB: fn.Fabrics):
8
9     # Establish a connection to the
10    connection = sqlDB.connect()
11    cursor = connection.cursor()
12
13
14    logging.info("Adding attendee ..")
15    # Create the table if it doesn't
16    create_table_query = """
17        IF OBJECT_ID(N'dbo.attendees',
18            [session_id] bigint NOT
19            [Attendee] NVARCHAR(250)
20            [Status] NVARCHAR(50) NO
21            );
22
23        ...
24
25        cursor.execute(create_table_quer
26
27        # Insert data into the table
28        # insert_query = "INSERT INTO db
29        merge_query = """
30            MERGE dbo.attendees AS target
31            USING (SELECT ? AS session_id
32            ON target.session_id = source.session_id
33            WHEN MATCHED THEN
34                UPDATE SET Status = ?
35            WHEN NOT MATCHED THEN
36                INSERT (session_id, Attendee, Status)
37                VALUES (?, ?, ?);

```

29

After it's finished you can see the output and logs again. We can see that we added some data to the attendee table.

The screenshot shows the Azure Data Studio interface with the following details:

- Left Sidebar:** Shows the workspace structure with a tree view of "udf_fabcon_scheduler".
- Top Bar:** Includes tabs for Home, Edit, Language (Python), Generate invocation code, Manage connections, Library management, Open in VS Code, and Publish.
- Middle Area:** A code editor window titled "Functions explorer" containing Python code for a User Defined Function (UDF). The code imports `fabric.functions` and `logging`, defines a UDF `save_to_agenda`, and creates a table `dbo.attendees` if it doesn't exist. It then inserts data into the table using a MERGE statement.

```

1 import fabric.functions as fn
2 import logging
3
4 udf = fn.UserDataFunctions()
5
6 @udf.connection(argName="sqlDB", alias="@udf.function()")
7 def save_to_agenda(sqlDB: fn.Fabrics
8
9     # Establish a connection to the
10    connection = sqlDB.connect()
11    cursor = connection.cursor()
12
13    logging.info("Adding attendee ..")
14    # Create the table if it doesn't
15    create_table_query = """
16        IF OBJECT_ID(N'dbo.attendees'
17        CREATE TABLE dbo.attendees (
18            [session_id] bigint NOT
19            [Attendee] NVARCHAR(250)
20            [Status] NVARCHAR(50) NO
21            );
22
23        ...
24
25        cursor.execute(create_table_quer
26
27        # Insert data into the table
28        # insert_query = "INSERT INTO db
29        merge_query = """
30            MERGE dbo.attendees AS targe
31            USING (SELECT ? AS session_i
32            ON target.session_id = src.s
33            WHEN MATCHED THEN
34                UPDATE SET Status = ?
35            WHEN NOT MATCHED THEN
36                INSERT (session_id, Atte
VALUES (?, ?, ?);
```
- Right Area:** A "Test" tab showing the results of the function execution. It includes parameters (sessionid: 10, Attendee: me me me, Status: Definitely), a "Test" button, and a summary section. The status is "Succeeded" with the message "Attendee table was created (if necessary) and data was added to this table". The logs section shows "Adding attendee ..." and "Attendee was added/modifed".
- Bottom Status Bar:** Shows "Test session active" and "Last published: 09/13/2025 01:51:13 PM".

Verify the data in the SQL database

30

Let's go see the data in the SQL database. Go back to the SQL database by clicking on its icon on the left, or by going back to the workspace and clicking on it there.

The screenshot shows the Power BI desktop application. On the left, the 'fabcon_sessions - GSMF 069' workspace is selected. In the center, a code editor displays a Python script for creating a table and inserting data into the 'attendees' table in a SQL database. On the right, a 'Test' results pane shows the execution status and output. The status is 'Succeeded' with the message: 'Attendee table was created (if necessary) and data was added to this table'. The logs pane shows the command executed: 'Adding attendee ...' and 'Attendee was added/modified'.

```
1 import fabric.functions as fn
2 import logging
3
4 udf = fn.UserDataFunctions()
5
6 @udf.connection(argName="sqlDB", alias="udf.function()")
7 def save_to_agenda(sqlDB: fn.FabricSQL):
8     # Establish a connection to the connection = sqlDB.connect()
9     cursor = connection.cursor()
10
11     logging.info("Adding attendee ..")
12     # Create the table if it doesn't
13     # create_table_query = """
14     #     IF OBJECT_ID(N'dbo.attendees',
15     #                 'TABLE') IS NOT NULL
16     #         DROP TABLE dbo.attendees;
17     #     CREATE TABLE dbo.attendees (
18     #         [session_id] bigint NOT
19     #         [Attendee] NVARCHAR(250)
20     #         [Status] NVARCHAR(50) NO
21     #         );
22
23     #     cursor.execute(create_table_query)
24
25     # Insert data into the table
26     # insert_query = "INSERT INTO db
27     #     merge_query = """
28     #         MERGE dbo.attendees AS target
29     #             USING (SELECT ? AS session_id
30     #                   ON target.session_id = src.session_id
31     #                   WHEN MATCHED THEN
32     #                       UPDATE SET Status = ?
33     #                   WHEN NOT MATCHED THEN
34     #                       INSERT (session_id, Attendee, Status)
35     #                           VALUES (?, ?, ?);"
36
```

31

Click on the attendees table to see a preview of the data. If you don't see anything yet, click on the refresh button in the Home ribbon or next to the Tables folder. It's possible you might still not see it in the preview (sometimes it takes a few minutes to show changes).

The screenshot shows the Power BI Data Explorer. The 'fabcon_sessions' workspace is selected. The 'Tables' folder is expanded, showing the 'attendees' table. A circular highlight is placed over the 'attendees' table icon in the 'Tables' folder. To the right, a 'Data preview - attendees' pane shows a table with three columns: 'session_id', 'Attendee', and 'Status'. The preview is set to show 1000 rows.

session_id	Attendee	Status
ABC	Attendee 1	Active
ABC	Attendee 2	Active
ABC	Attendee 3	Active
ABC	Attendee 4	Active
ABC	Attendee 5	Active
ABC	Attendee 6	Active
ABC	Attendee 7	Active
ABC	Attendee 8	Active
ABC	Attendee 9	Active
ABC	Attendee 10	Active

32

If the preview doesn't show anything yet, you can see it by writing a SQL query. Click "New Query."

The screenshot shows the Databricks interface with the 'fabcon sessions' database selected. The left sidebar shows various databases and workspaces. The main area has a search bar at the top. Below it, the 'Home' tab is selected. A prominent orange circle highlights the 'New Query' button in the top navigation bar. To its right are other buttons like 'Get data', 'Templates', 'Open in', 'New API for GraphQL', 'Performance summary', and 'Copilot'. The central workspace shows an 'Explorer' sidebar with 'fabcon sessions' expanded, showing 'dbo' and 'attendees' under 'Tables'. A 'SQL query 1' tab is open, displaying a table named 'attendees' with columns 'session_id', 'Attendee', and 'Status'. A message at the bottom says 'Showing 1000 rows'.

33

Enter "SELECT * FROM attendees" and then click the "Run" button.

This screenshot continues from the previous one. The 'New Query' button has been clicked, creating a new query tab titled 'SQL query 2'. The 'Run' button in the toolbar of this tab is highlighted with an orange circle. The query 'SELECT * FROM attendees' is typed into the editor. The rest of the interface is identical to the previous screenshot, including the sidebar, top navigation, and the preview of the 'attendees' table.

34

You should now see one row. You populated the table by running the UDF!

The screenshot shows the Databricks interface with the following details:

- Left Sidebar (Explorer):** Shows a tree view of databases, tables, and queries. The 'attendees' table under 'fabcon sessions' is selected.
- Top Bar:** Contains tabs for 'SQL query 1' and 'SQL query 2'. The 'attendees' table is selected in 'SQL query 1'.
- Code Editor:** Displays the SQL query: `1 SELECT * FROM attendees`.
- Results Tab:** Active tab, showing the output of the query. The table has three columns: session_id, Attendee, and Status. One row is present: session_id 10, Attendee 'me me me', and Status 'Definitely'.

35

Now go back to the UDF so we can test the delete function. You'll need to use the same session_id and Attendee so it can delete the correct row.

The screenshot shows the Databricks interface with the following details:

- Left Sidebar (Copilot):** Shows a tree view of databases, tables, and queries. A specific entry, 'udf.fabcon_scheduler - GSFM 069', is highlighted with a red circle.
- Top Bar:** Contains tabs for 'SQL query 1' and 'SQL query 2'. The 'attendees' table is selected in 'SQL query 1'.
- Code Editor:** Displays the SQL query: `1 SELECT * FROM attendees`.
- Results Tab:** Active tab, showing the output of the query. The table has three columns: session_id, Attendee, and Status. One row is present: session_id 10, Attendee 'me me me', and Status 'Definitely'.

Test the second function

36

If you didn't close the test window, you can simply scroll to the top and select the delete function in the Function name dropdown. This should leave in the parameters you already had. If you did close it, just hover over the delete function and click the test tube to test. Then fill in the same sessionid and Attendee values you used for the first test and click the "Test" button.

The screenshot shows the Power BI Editor interface. On the left, the 'Functions explorer' pane lists two functions: 'fx save_to_agenda' and 'fx delete_from_agenda'. The main area displays Python code for the 'fx delete_from_agenda' function. The code performs a database operation to remove an attendee from a table named 'Attendee'. It uses a cursor to execute a query and logs the action. The 'Test' dialog is open on the right, with the 'Function name' dropdown set to 'delete_from_agenda'. The 'Parameters' section shows 'sessionid' with value '10' and 'Attendee' with value 'me me me'. A large orange circle highlights the 'Test' button, which is also highlighted with a green border. Below the test dialog, there are sections for 'Output (str)' and 'Logs'.

```
44     status,
45     sessionid,
46     attendee,
47     # IN
48     status
49     )
50     cursor.execute(merge_query, merg
51     logging.info("Attendee was added"
52     # Commit the transaction
53     connection.commit()
54
55     # Close the connection
56     cursor.close()
57     connection.close()
58     return "Attendee table was creat
59
60     @udf.connection(argName="sqlDB", alias
61     @udf.function()
62     def delete_from_agenda(sqlDB: fn.Fab
63
64     data = (sessionid, attendee)
65
66     # Establish a connection to the
67     # connection = sqlDB.connect()
68     cursor = connection.cursor()
69
70     logging.info("Removing attendee")
71     # Insert data into the table
72     insert_query = "DELETE FROM dbo.
73     cursor.execute(insert_query, dat
74     logging.info("Attendee was reme
75
76     # Commit the transaction
77     connection.commit()
78
79     # Close the connection
80     cursor.close()
```

Test

Use this to test your functions locally in Develop mode.

Function name

delete_from_agenda

Parameters

Type: int

sessionid

10

Type: str

Attendee

me me me

Test

Output (str)

Logs

● Test session active Last published: 09/13/2025 01:51:13 PM

37 The output and logs should tell you that you deleted a row.

The screenshot shows the Power BI Data Studio interface for the workspace "udf_fabcon_scheduler".

Left Sidebar:

- Home
- Create
- Browse
- OneLake catalog
- Workspaces
- GSMF 069
- udf_fabcon_scheduler (selected)
- fabcon_sessions
- Transform FabCon ...
- fabcon_sessi on_info
- fabcon_sessi on_info
- ...
- Power BI

Top Bar:

- Home Edit
- Language Python
- Generate invocation code
- Manage connections
- Library management
- Open in VS Code
- Publish

Middle Section:

This user data functions item has unpublished changes. Make sure to add all necessary libraries and connections before publishing.

Functions explorer:

- fx save_to_agenda
- fx delete_from_agenda

Code Editor:

```
    status,
    sessionid,
    Attendee,
    Status
)
cursor.execute(merge_query, merg
logging.info("Attendee was added
)

# Commit the transaction
connection.commit()
)

# Close the connection
cursor.close()
connection.close()
return "Attendee table was creat

@udf.connection(argName="sqlDB",alias
@udf.function()
def delete_from_agenda(sqlDB: fn.Fab
data = (sessionid, Attendee)

# Establish a connection to the
connection = sqlDB.connect()
cursor = connection.cursor()

logging.info("Removing attendee")
# Insert data into the table
insert_query = 'DELETE FROM dbo.
cursor.execute(insert_query, dat
logging.info("Attendee was remeo

# Commit the transaction
connection.commit()

# Close the connection
```

Test Results:

Test

delete_from_agenda

Parameters:

Type: int
sessionid: 10

Type: str
Attendee: me me me

Status: Succeeded

Output (str): Attendee was removed

Logs:

Removing attendee
Attendee was remeoved

38

Go back to the SQL database again and we'll make sure the row is gone.

The screenshot shows a SQL database interface with a sidebar containing various database objects like 'OneLake catalog', 'Workspaces', and 'fabcon_sessions - GSMF 069'. The main area displays a Python script named 'delete_from_agenda' with the following code:

```

44     sessionid,      # IN
45     Attendee,       # ..
46     Status         # ..
47   )
48   cursor.execute(merge_query, merg
49   logging.info("Attendee was added"
50
51   # Commit the transaction
52   connection.commit()
53
54   # Close the connection
55   cursor.close()
56   connection.close()
57   return "Attendee table was creat
58
59 @udf.connection(argName="sqlDB", alias
60 @udf.function()
61 def delete_from_agenda(sqlDB: fn.Fab
62
63   data = (sessionid, Attendee)
64
65   # Establish a connection to the
66   connection = sqlDB.connect()
67   cursor = connection.cursor()
68
69   logging.info("Removing attendee"
70   # Insert data into the table
71   insert_query = "DELETE FROM dbo.
72   cursor.execute(insert_query, dat
73   logging.info("Attendee was reme
74
75   # Commit the transaction
76   connection.commit()
77
78   # Close the connection

```

To the right, a 'Test' window is open with the following parameters:

- Type: int sessionid 10
- Type: str Attendee me me me

The 'Status' section shows 'Succeeded'. The 'Output (str)' section contains the message 'Attendee was removed'. The 'Logs' section shows the command 'Removing attendee' and the confirmation 'Attendee was removed'.

39

Rerun the SQL query.

The screenshot shows a SQL database interface with a sidebar containing various database objects. The main area displays the results of a SQL query:

```

1  SELECT * FROM attendees

```

The results table has three columns: session_id, Attendee, and Status. The single row returned is:

	session_id	Attendee	Status
1	10	me me me	Definitely

40 The row is gone! Back to the empty table. Navigate back to the UDF.

The screenshot shows the Azure Data Studio interface. On the left, the Explorer sidebar displays the database schema, including the 'dbo' schema which contains the 'attendees' table. The main area shows two SQL queries. The first query, 'SQL query 1', contains the command 'SELECT * FROM attendees'. The second query, 'SQL query 2', is currently selected and displays the results of the previous query. The results table has three columns: 'session_id', 'Attendee', and 'Status'. The entire results table is highlighted with an orange border. The status bar at the bottom indicates 'Succeeded (0 sec 698 ms)' and 'Copilot completions: Ready'.

session_id	Attendee	Status
ABC		

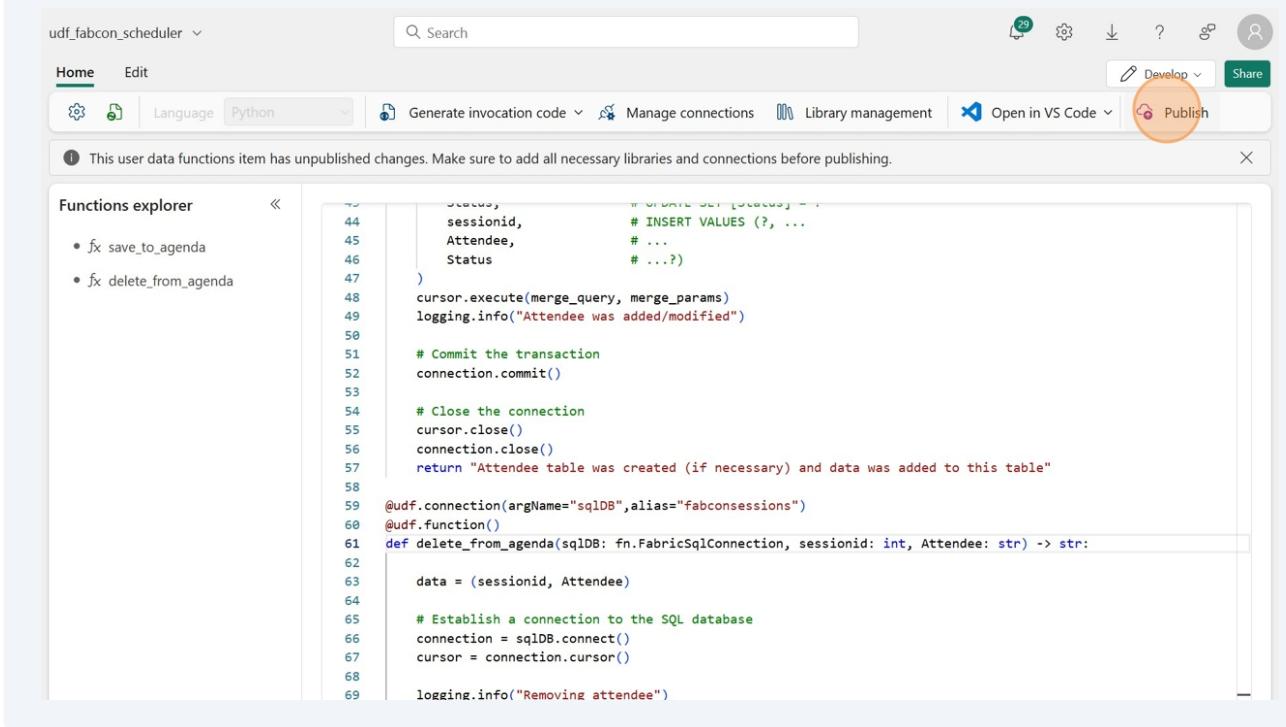
41 Close the test pane if you've still got it open.

The screenshot shows the Databricks workspace interface. On the left, the sidebar lists notebooks like 'udf_fabcon_scheduler', 'fabcon_sessions', 'Transform', 'fabcon_session_info', and 'fabcon_session_info'. The main area displays Python code for a user-defined function (UDF). The code uses pandas DataFrames and SQLAlchemy to interact with a database. The 'Test' pane on the right shows a successful execution of the UDF with the specified parameters. A red circle highlights the close button in the top right corner of the Test pane.

```
44     status,
45     sessionid,      # IN
46     Attendee,       # ..
47     Status          # ..
48 )
49 cursor.execute(merge_query, merge)
50 logging.info("Attendee was added")
51
52 # Commit the transaction
53 connection.commit()
54
55 # Close the connection
56 cursor.close()
57 connection.close()
58 return "Attendee table was created"
59
60 @udf.connection(argName="sqlDB", alias="udf.function()")
61 def delete_from_agenda(sqlDB: fn.Fab):
62
63     data = (sessionid, Attendee)
64
65     # Establish a connection to the
66     # connection = sqlDB.connect()
67     cursor = connection.cursor()
68
69     logging.info("Removing attendee")
70     # Insert data into the table
71     insert_query = "DELETE FROM dbo."
72     cursor.execute(insert_query, data)
73     logging.info("Attendee was removed")
74
75     # Commit the transaction
76     connection.commit()
77
78     # Close the connection
```

Publish the User data function

42 Click "Publish" to publish the function.



The screenshot shows the Azure Functions portal interface. The top navigation bar includes 'Home', 'Edit', 'Generate invocation code', 'Manage connections', 'Library management', 'Open in VS Code', and a 'Publish' button, which is circled in red. A status message at the top left says, 'This user data functions item has unpublished changes. Make sure to add all necessary libraries and connections before publishing.' The main area is titled 'Functions explorer' and contains two items: 'fx save_to_agenda' and 'fx delete_from_agenda'. Below these are code snippets for each function. The 'delete_from_agenda' function code is as follows:

```
    Status,          "# UPDATE SET [Status] = ...  
44    sessionid,      # INSERT VALUES (?, ...  
45    Attendee,       # ...  
46    Status          # ...?)  
47 )  
48 cursor.execute(merge_query, merge_params)  
49 logging.info("Attendee was added/modified")  
50  
51 # Commit the transaction  
52 connection.commit()  
53  
54 # Close the connection  
55 cursor.close()  
56 connection.close()  
57 return "Attendee table was created (if necessary) and data was added to this table"  
58  
59 @udf.connection(argName="sqlDB",alias="fabconssessions")  
60 @udf.function()  
61 def delete_from_agenda(sqlDB: fn.FabricSqlConnection, sessionid: int, Attendee: str) -> str:  
62  
63     data = (sessionid, Attendee)  
64  
65     # Establish a connection to the SQL database  
66     connection = sqlDB.connect()  
67     cursor = connection.cursor()  
68  
69     logging.info("Removing attendee")
```

43 You'll see a message that the function is being published.

```
import fabric.functions as fn
import logging

udf = fn.UserDataFunctions()

@udf.connection(argName="sqlDB", alias="fabconsessions")
@udf.function()
def save_to_agenda(sqlDB: fn.FabricSqlConnection, sessionid: int, Attendee: str, Status: str) -> str:

    # Establish a connection to the SQL database
    connection = sqlDB.connect()
    cursor = connection.cursor()

    logging.info("Adding attendee ... ")
    # Create the table if it doesn't exist
    create_table_query = """
        IF OBJECT_ID(N'dbo.attendees', N'U') IS NULL
            CREATE TABLE dbo.attendees (
                [session_id] bigint NOT NULL,
                [Attendee] NVARCHAR(256) NOT NULL,
                [Status] NVARCHAR(50) NOT NULL
            );
    """
    cursor.execute(create_table_query)

    # Insert data into the table
    # insert_query = "INSERT INTO dbo.attendees (session_id, Attendee, Status) VALUES (?, ?, ?);"
    merge_query = """
        MERGE dbo.attendees AS target
        USING (SELECT ? AS session_id, ? AS Attendee) AS src
        ON target.session_id = src.session_id AND target.Attendee = src.Attendee
        WHEN MATCHED THEN
            UPDATE SET Status = ?
        WHEN NOT MATCHED THEN
            INSERT (session_id, Attendee, Status)
            VALUES (?, ?, ?);
    """
    """
    cursor.execute(insert_query, (sessionid, Attendee, Status))
    cursor.execute(merge_query, (sessionid, Attendee, Status))
    """

cursor.execute(create_table_query)

# Insert data into the table
# insert_query = "INSERT INTO dbo.attendees (session_id, Attendee, Status) VALUES (?, ?, ?);"
merge_query = """
    MERGE dbo.attendees AS target
    USING (SELECT ? AS session_id, ? AS Attendee) AS src
    ON target.session_id = src.session_id AND target.Attendee = src.Attendee
    WHEN MATCHED THEN
        UPDATE SET Status = ?
    WHEN NOT MATCHED THEN
        INSERT (session_id, Attendee, Status)
        VALUES (?, ?, ?);
"""
"""
cursor.execute(insert_query, (sessionid, Attendee, Status))
cursor.execute(merge_query, (sessionid, Attendee, Status))
"""

udf.publish()
```

Test session active Publishing in progress

44

Go back to the workspace and you'll see that you now have a User Data Function.

The screenshot shows a workspace interface with a sidebar on the left containing various project and file icons. The main area displays a table of items with the following columns: Name, Type, Task, Owner, Refreshed, Next refresh, Endorsement, Sensitivity, and Included in app. One item, 'udf_fabcon_scheduler', is highlighted with an orange border. The table data is as follows:

Name	Type	Task	Owner	Refreshed	Next refresh	Endorsement	Sensitivity	Included in app
fabcon sessions	SQL database	—	GSMF Dev...	—	—	—	—	—
fabcon sessions	SQL analysis	—	GSMF Dev...	—	—	—	—	—
fabcon_session_info	Lakehouse	—	GSMF Dev...	—	—	—	—	—
fabcon_session_info	SQL analysis	—	GSMF Dev...	—	—	—	—	—
Transform FabCon Session Info	Dataflow	—	GSMF Dev...	—	—	—	—	—
udf_fabcon_scheduler	User data	—	GSMF Dev...	—	—	—	—	—