

# Step 1: Load Festival Data

For our lab, we will use some made-up festival data. To support this, we will need to load multiple tables. In the Data folder you will see all the files needed.

Log into your Snowflake Account and create a space for these tables to live through the UI. Let's create a database we will call it SI\_EVENTS\_HOL but you can use any database/schema you have write privileges on.

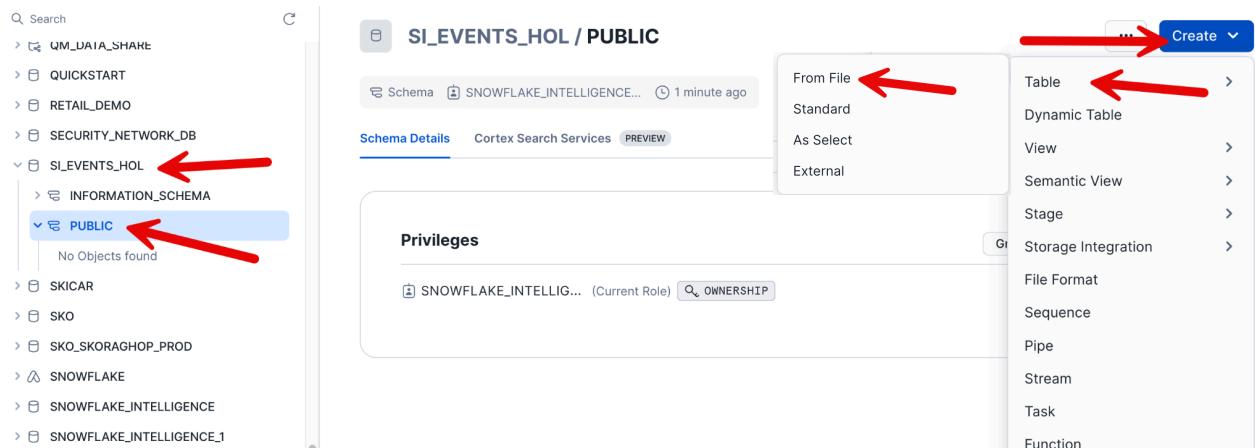
The screenshot shows the Snowflake web interface. On the left, a sidebar menu includes 'Create', 'Home', 'Search', 'Projects', 'Data' (which is selected and highlighted in blue), 'Databases' (also highlighted with a red arrow), 'Data Products', 'AI & ML', 'Monitoring', and 'Admin'. At the bottom of the sidebar, there is a user icon and the text 'SNOWFLAKE\_INTELLIGENCE\_ADMIN\_RL'. The main area is titled 'Databases' and shows a list of 55 existing databases. A red arrow points to the '+ Database' button at the top right of this list. Below this, a modal window titled 'New Database' is open, showing the text 'Creating as SNOWFLAKE\_INTELLIGENCE\_ADMIN\_RL'. It has a 'Name' field containing 'SI\_Events\_HOL' with a red arrow pointing to it, and a 'Save' button with a red arrow pointing to it.

Let's create the tables we need for this lab. There are a total of 4 tables needed for this lab, and they can all be created the same way by creating a table from a file in Snowflake. Please repeat the following steps for all four tables:

1. Customers
2. Contracts

3. Ticket Sales
4. Events

To load a table from a file in Snowsight, go into the database, go to the schema, and click the create button on the top right to create a table from a file:



Name each table the same name as the file you unzipped:

**Load Data into Table**

SNOWFLAKE\_INTELLIG...

SI\_EVENTS\_HOL.PUBLIC.TICKET\_SALES

ticket\_sales.csv - 1.6MB

Browse

Select or create a database and schema

Schema SI\_EVENTS\_HOL.PUBLIC ▾ + Database

Select or create a table

+ Create new table

Name

ticket\_sales

Cancel Back Next

Open the View Options section in the file format area and make sure that the header is the first line of the document:

**Load Data into Table**

contracts.csv → SI\_EVENTS\_HOL.PUBLIC.CONTRACTS

File format

Delimited Files (CSV or TSV)

Select existing or create in [Worksheets](#) ↗

Learn more about format-specific configurations in [Snowflake Docs](#) ↗

**View options** ↙

**Edit Schema**

3 Columns

<input checked="" type="checkbox"/> DATA TYPE	COLUMN NAME
<input checked="" type="checkbox"/> VARCHAR	DOCUMENT_TITLE

Load Data into Table

contracts.csv → SI\_EVENTS\_HOL.PUBLIC.CONTRACTS

File format: Delimited Files (CSV or TSV)

Header: First line contains header

Field delimiter: No header

Trim space: Skip first line

Field optionally enclosed by: Custom

Replace invalid characters: True (default)

Edit Schema:

DATA TYPE	COLUMN NAME	COLUMN DATA
VARCHAR	DOCUMENT_TITLE	Artist_Agreement_The_Algorhythms, Ar...
VARCHAR	URL	https://music-fest.sharepoint.com/doc...
VARCHAR	TEXT	The Algorhythms will perform on the M...

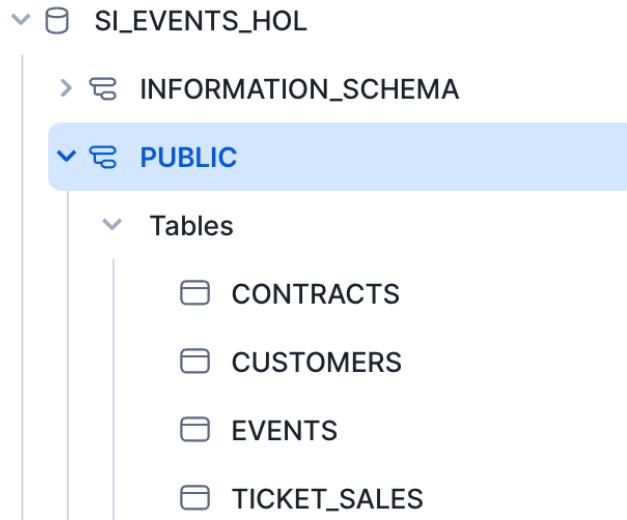
Successfully Loaded Data

10 rows were successfully inserted into the table.

Query Data

Done

Repeat this for all 4 files so you have all 4 tables needed for the lab. When you are complete, you should see the following under tables in the public schema:



## Step 2: Create a Semantic View

Let's now build a more intelligent agent that analyzes the data within Snowflake by utilizing our semantic layer.

Then we can create a semantic view that we can use in an agent. Let's do that through Snowflake AI & ML Studio using Cortex Analyst:

Go ahead and select create new:

We can choose the SNOWFLAKE\_INTELLIGENCE database and CONFIG schema we created earlier, as well as the stage of SEMANTIC\_MODELS:

Name: Music Festival

Description: This has information on music festivals, including locations, events, and ticket sales

Create Semantic View

SYSADMIN • DEMO\_WH (X-Small)

Getting started

Select tables

Select columns

**Getting started**

As you start creating this semantic view, make sure you're focused on a well-defined and scoped use case.

Location to store

DEMO.SI ▾

Name

MUSIC\_FESTIVAL

Description (optional)

This has information on music festivals, including locations, events, and ticket sales

Cancel

Next

Let's select our tables that we imported at the beginning of the lab:

Create Semantic Model

Snowflake\_Intelligence\_Admi...

Getting started

**Select tables**

Select columns

### Select tables

All   Selected 4

Search

SI\_EVENTS\_HOL

INFORMATION\_SCHEMA

PUBLIC

4 Tables

CONTRACTS

CUSTOMERS

EVENTS

TICKET\_SALES

The screenshot shows the 'Create Semantic Model' interface. On the left, there are three radio button options: 'Getting started' (checked), 'Select tables' (selected and highlighted in blue), and 'Select columns'. The right panel is titled 'Select tables' and shows a tree view of database schemas. Under 'SI\_EVENTS\_HOL', the 'PUBLIC' schema is expanded, revealing four tables: CONTRACTS, CUSTOMERS, EVENTS, and TICKET\_SALES. Each table has a checkbox next to it, and all four are checked. A 'Selected' tab at the top indicates 4 selected items.

## View the Results

Then we can select the columns to choose. Let's just select all columns by choosing the top checkbox:

When we click 'Create and Save' it will generate a starter semantic model for us:

Getting started Select tables Select columns

## Select columns

Select only the columns required to answer the business questions your users want to ask.

Columns (12 selected) Expand all

Search columns

- >   DEMO.SI.CUSTOMERS
- >   DEMO.SI.EVENTS
- >   DEMO.SI.TICKET\_SALES

**Add sample values to the semantic view**

A few real values from each column helps Cortex Analyst give better answers. [Learn more](#)

**Add descriptions to the semantic view**

We'll use AI to figure out sensible descriptions to add in, saving you time.

[Cancel](#)[Previous](#)[Create and Save](#)

The screenshot shows the Cortex Studio interface with the 'Semantic Model' tab selected. The main content area displays a semantic model named 'music\_festival'. It includes a description: 'This has information on music festivals, including locations, events, and ticket sales.' Below this is a 'Custom Instructions' section with an 'Edit' button. The 'Logical tables' section contains four entries, each with an 'Edit' button and a delete icon. One entry is expanded, showing a question about what a logical table is and a note that 4 of 4 logical tables are missing synonyms. The 'CONTRACTS' section is also visible, showing a table named 'SI\_EVENTS\_HOL.PUBLIC.CONTRACTS' with 3 dimensions and 0 time dimensions.

Cortex Studio Cortex Analyst PREVIEW SNOWFLAKE\_INTELLIGENCE\_ADMIN...

Semantic Model Monitoring

SEMANTIC\_MODELS > music\_festival.yaml <> Edit YAML

**music\_festival** Edit

This has information on music festivals, including locations, events, and ticket sales.

**Custom Instructions** Edit

**Logical tables** 4 +

- ⓘ What is a logical table?  
A logical table can be thought of as a simple view over a physical database table or view. Each logical table is a collection of logical columns, which are categorized into Dimensions, Time Dimensions, and Facts. [Learn more](#)
- ⓘ 4 of 4 logical tables missing synonyms  
More synonyms can improve the quality of responses from Cortex Analyst.

**CONTRACTS** Edit ⚡

SI\_EVENTS\_HOL.PUBLIC.CONTRACTS

> Dimensions 3 +

> Time Dimensions 0 +

We can see that it has already created a description as well as some other details, like synonyms. Let's go ahead and add 'show' and 'concert' to the list of synonyms for EVENT\_NAME in the EVENTS table:

**Expression** ⓘ

+ Add column

EVENT\_NAME

Dimension name

EVENT\_NAME

Generate Fields

Data type

VARCHAR(16777216)

Dimension description (optional)

The name of the event that occurred.

Synonyms (optional)

le, event\_description, event\_label, event\_heading, event\_caption, event\_header, show, concert

Connect Cortex Search (optional) ⓘ

+ Search Service

Sample values

+ Value

Instead Fest

×

## Test the Model

I can test this model right now by going to the side window and putting in the following prompt:  
*What are the different events in Europe*

Then it will go against my model and then write SQL, and then execute that SQL against my model:

The screenshot shows the Cortex Studio interface. On the left, a 'Model' configuration for 'music\_festival.yaml' is displayed. It includes fields for 'Expression' (EVENT\_NAME), 'Dimension name' (EVENT\_NAME), 'Data type' (VARCHAR(16777216)), 'Dimension description (optional)' (The name of the event that occurred.), 'Synonyms (optional)' (event\_title, event\_description, event\_label, event\_heading, event\_caption, event\_header, show,), 'Connect Cortex Search (optional)', and 'Sample values' (Instead Fest, Ok Fest, Difficult Fest). On the right, a semantic search interface shows a query 'User What are the different events in Europe'. The results are a table of events in Europe from January to June 2025:

EVENT_NAME	START_DATE	END_DATE
Instead Fest	2025-06-13	2025-06-
Ok Fest	2025-03-12	2025-03-
Difficult Fest	2025-02-05	2025-02-
Wall Fest	2025-01-23	2025-01-
Oil Fest	2025-01-09	2025-01-

Below the table, a query editor shows the executed physical query:

```

SELECT
    event_name,
    MIN(event_date) AS start_date,
    MAX(event_date) AS end_date
  
```

Buttons for 'Run' and 'Semantic query' are visible.

## Move Columns

The model incorrectly identified EVENT\_ID as a fact due to its numerical data type, but it is actually a dimensional column. This can be easily corrected by moving it to the dimensions section in the edit menu. It is crucial to review and adjust the model after Snowflake's initial column categorization.

**TICKET\_SALES**

SI\_EVENTS\_HOL.PUBLIC.TICKET\_SALES

Dimensions 1 +

A TICKET\_TYPE

The type of ticket purchased, which can be one of three options: General, VIP, or Early Bird, indicating the level of access or perks associated with the ticket.

TICKET\_TYPE · VARCHAR(16777216)

Time Dimensions 1 +

Facts 4 +

# TICKET\_ID

Unique identifier for each ticket sale transaction.

TICKET\_ID · NUMBER(38,0)

# EVENT\_ID

Unique identifier for the event associated with the ticket sale.

EVENT\_ID · NUMBER(38,0)

# CUSTOMER\_ID

Unique identifier for the customer who purchased the ticket.

CUSTOMER\_ID · NUMBER(38,0)

We will need to do this for the following incorrectly identified columns:

1. TICKET\_SALES
  - a. TICKET\_ID - need to move from fact to dimension
  - b. EVENT\_ID - need to move from fact to dimension
  - c. CUSTOMER\_ID - need to move from fact to dimension
2. CUSTOMERS
  - a. CUSTOMER\_ID - need to move from fact to dimension

## Define Table Relationships

We then need to define our relationships with the other tables. By default, no relationships are created, but they are needed for more complex queries that span multiple tables.

Our first relationship we will define is TICKET\_SALES to EVENT. For every EVENT, there are many ticket sales, and the joining column is EVENT\_ID. Let's define a relationship that represents that:

Relationship Name: TicketSales\_to\_Events  
Join Type: Inner  
Relationship type: many\_to\_one  
Left Table: Ticket\_Sales  
Right Table: Events  
Relationship Columns: EVENT\_ID and EVENT\_ID

The screenshot shows a modal dialog titled "Add relationship". At the top left, there is a button labeled "Relationships" with a value of "0". To the right of this button is a large red arrow pointing towards a blue "Add" button at the top right of the dialog. The main area of the dialog contains fields for defining the relationship:

- Relationship name:** TicketSales\_to\_Events
- Join type:** inner
- Relationship type:** many\_to\_one
- Left table:** TICKET\_SALES
- Right table:** EVENTS
- Relationship columns:** EVENT\_ID (selected in both Left column and Right column dropdowns)

At the bottom right of the dialog are two buttons: "Cancel" and "Add".

Let's add a second relationship of customers to tickets:

Relationship Name: TicketSales\_to\_Customers  
Join Type: Inner  
Relationship type: many\_to\_one  
Left Table: Customers  
Right Table: Ticket\_Sales  
Relationship Columns: CUSTOMER\_ID and CUSTOMER\_ID

Relationship name

Join type

Relationship type

Left table

Right table

Relationship columns

Left column

Right column

+ Columns

Cancel

Save

- Relationships** 2 ← +
- 
- TicketSales\_to\_Events** Edit
- TICKET\_SALES . EVENT\_ID — EVENTS . EVENT\_ID
- Join: inner · Relationship: many\_to\_one
- 
- Customer\_to\_TicketSales** Edit
- CUSTOMERS . CUSTOMER\_ID — TICKET\_SALES . CUSTOMER\_ID
- Join: inner · Relationship: many\_to\_one

Now we have two relationships set up in our semantic model. Let's go ahead and test them using the prompt on the right side. We can ask this: *How many ticket sales were there for Difficult Fest?*

→ music\_festival ◌

User

How many ticket sales were there for Difficult Fest?

Cortex Analyst

Request ID



This is our interpretation of your question:

How many ticket sales were there for Difficult Fest over the entire available time period?

START_DATE	END_DATE	TICKET_SALE
2025-01-01	2025-02-05	758



✓ Physical query (executed) · Semantic query

✖

```
SELECT
    MIN(ts.sale_date) AS start_date,
    MAX(ts.sale_date) AS end_date,
    COUNT(*) AS ticket_sales
FROM
    ticket_sales AS ts
    INNER JOIN events AS e ON ts.event_id =
WHERE
    e.event_name = 'Difficult Fest'
    /* Generated by Cortex Analyst */
```

Enter prompt

▶ Run

We can see that it was able to utilize our join and see how many tickets were sold for that event by joining the two tables together we defined in our relationship.

We can also ask it to ‘Count the total customers by customer region that went to Difficult Fest’, and this should span all of our tables in our semantic model and give us values:

→ music\_festival

### User

Count the total customers by customer region that went to Difficult Fest

### Cortex Analyst

Request ID



This is our interpretation of your question:

What is the count of unique customers by customer region who attended Difficult Fest over the entire available time period?

REGION	CUSTOMER_COUNT	EVENT_START
South America	161	2025-02-0
Europe	161	2025-02-0
Africa	155	2025-02-0
North America	144	2025-02-0
Asia	137	2025-02-0

Physical query (executed) · Semantic query



```
WITH event_customers AS (
  SELECT
    DISTINCT c.customer_id,
    c.region
  FROM
    customers AS c
    INNER JOIN ticket_sales AS ts ON c.cust
    INNER JOIN events AS e ON ts.event_id =
  WHERE
    e.event_name = 'Difficult Fest'
```

Excellent! This functionality is operating as anticipated.

## Verify the Queries

Since these queries returned the correct values, we can add them as verified queries, which will fuel our model's improvement.

→  music\_festival 

```
WITH unique_customers AS (
    SELECT
        DISTINCT c.region,
        c.customer_id
    FROM
        customers AS c
    INNER JOIN ticket_sales AS ts ON c.cus
    INNER JOIN events AS e ON ts.event_id
    WHERE
        e.event_name = 'Difficult Fest'
)
SELECT
    uc.region,
    COUNT(uc.customer_id) AS customer_count,
    MIN(ts.sale_date) AS start_date,
    MAX(ts.sale_date) AS end_date
FROM
    unique_customers AS uc
    LEFT JOIN ticket_sales AS ts ON uc.custo
GROUP BY
    uc.region
ORDER BY
    customer_count DESC NULLS LAST
/* Generated by Cortex Analyst */
```



+ Verified query

## Add verified query

Verified query name

Events in Europe

Question

What are the different events in Europe

Add this question to [onboarding questions](#)

Query

Semantic query

▶ Test

```
1  SELECT
2      MIN(ts.sale_date) AS start_date,
3      MAX(ts.sale_date) AS end_date,
4      COUNT(*) AS ticket_sales
5  FROM
6      ticket_sales AS ts
7      INNER JOIN events AS e ON ts.event_id = e.event_id
8  WHERE
9      e.event_name = 'Difficult Fest'
```

Only Semantic queries can be saved as verified queries.

[Learn more](#)

Cancel

Add

These will now show up under verified queries in our model definition:

## Verified Queries 2

+

- ✓ Count the total customers by customer region that went to Difficult Fest

Edit 

Difficult Fest Customers · By Kirsten Childs · May 29, 2025

```
WITH event_customers AS (
  SELECT
    DISTINCT c.customer_id,
    c.region
  FROM
    customers AS c
    INNER JOIN ticket_sales AS ts ON c.customer_id = ts.customer_id
    INNER JOIN events AS e ON ts.event_id = e.event_id
  WHERE
    e.event_name = 'Difficult Fest'
```

- ✓ How many ticket sales were there for Difficult Fest?

Edit 

Events in Europe · By Kirsten Childs · May 29, 2025

```
SELECT
  MIN(ts.sale_date) AS start_date,
  MAX(ts.sale_date) AS end_date,
  COUNT(*) AS num_ticket_sales
FROM
  ticket_sales AS ts
  INNER JOIN events AS e ON ts.event_id = e.event_id
WHERE
  e.event_name = 'Difficult Fest'
```

## Save the Model

When you are done, make sure you save the model:

The screenshot shows the Snowflake interface for managing semantic models. At the top, there are tabs for 'SNOWFLAKE\_INTELLIGENCE\_ADMIN...' and 'SNOWFLAKE\_INTELLIGENCE\_WH (...'. Below these are buttons for 'Open', 'Save', and '...'. The main area displays a semantic model named 'music\_festival' with the following SQL code:

```
SELECT
    DISTINCT c.region,
    c.customer_id
FROM
    customers AS c
INNER JOIN ticket_sales AS ts ON c.customer_id = ts.customer_id
INNER JOIN events AS e ON ts.event_id = e.event_id
WHERE
    e.event_name = 'Difficult Fest'
)
SELECT
    uc.region,
    COUNT(uc.customer_id) AS customer_count,
    MIN(ts.sale_date) AS start_date,
```

## Step 3: Create an Agent to Use the Semantic Model

Let's go back to the agents now and create a new Agent

The screenshot shows the 'AI & ML' section of the Snowflake interface. On the left, there is a sidebar with navigation links: '+ Create', 'Home', 'Search', 'Projects', 'Data', 'Data Products', 'AI & ML' (which has a red arrow pointing to it), 'Studio', 'Cortex Search', 'Features', 'Models', 'Evaluations', 'Document AI', 'Agents' (which has a red arrow pointing to it), and 'Snowflake Intelligence'. The main area is titled 'Agents' and shows '2 Agents'. It lists an agent named 'Claude' with the note: 'This agent is connected securely to Claude in Snowflake.' At the top right, there is a 'Documentation' link and a blue button labeled '+ Create agent' with a red arrow pointing to it.

We will use the following information:

Name: Music Festival Agent

Description: An agent that can return information about music festival events and ticket sales

**Create agent**

Platform integration

Create this agent for [Snowflake Intelligence](#)

Database and schema

SNOWFLAKE\_INTELLIGENCE.AGENTS

Agent object name

music\_festival

ⓘ API URL is based on the agent's name. Changing the name later may break the link./databases/SNOWFLAKE\_INTELLIGENCE/schemas/AGENTS/agents/music\_festival

Display name

Music Festival

CancelCreate agent

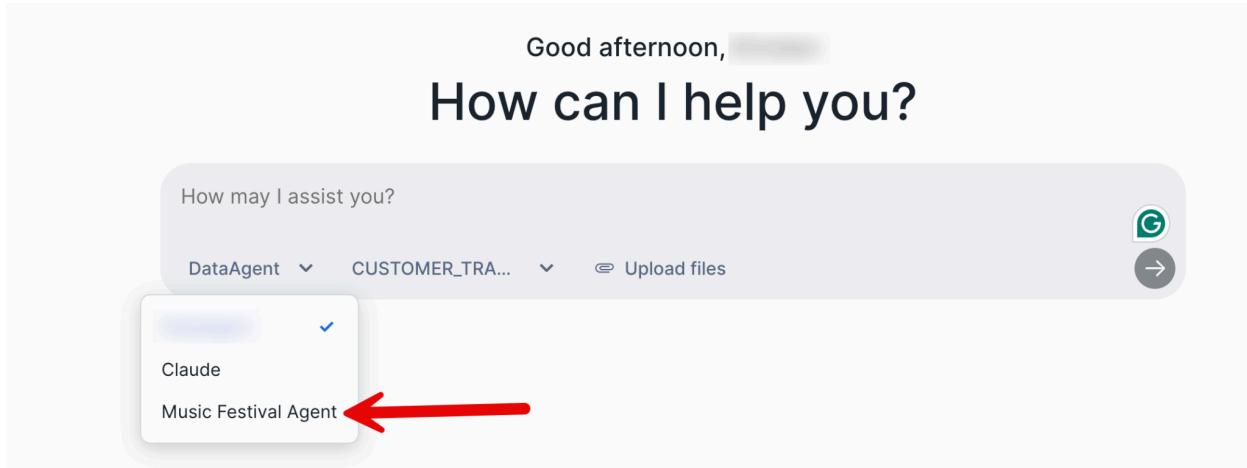
After those are filled in, go ahead and click Create Agent

Now click the agent and in the top right corner click Edit, then head on over to Tools on the left hand side and click + Add to add our Cortex Analyst. Give it a user friendly name and generate the description with cortex.

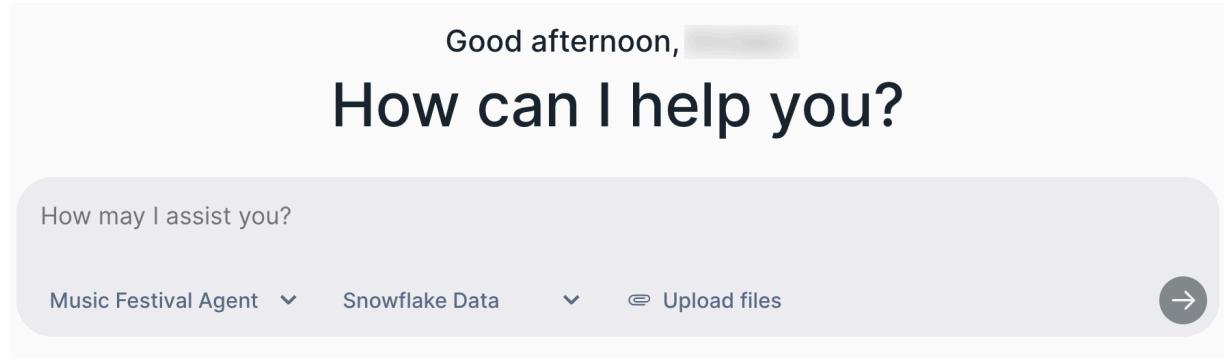
## Step 4: Test in Snowflake Intelligence

Let's go back to Snowflake Intelligence to see our new agent, which leverages our semantic model. Either click Snowflake Intelligence or go to your Snowflake Intelligence tab and refresh the page.

We can now see the Music Festival Agent.



We can also see that it shows our Snowflake Data, which is what we called our music festival.yml semantic file we created earlier:



Let's ask it a question: [\*Show me the top 5 events by tickets sold.\*](#)

Show me the top 5 events by tickets sold.

✓ Thinking completed ✓ 1 verified query

Show Details ▾

Based on the SQL results from the tool execution, here are the top 5 events by tickets sold:

1. Instead Fest: 4,514 tickets
2. Age Fest: 3,608 tickets
3. Least Fest: 3,287 tickets
4. Own Fest: 3,285 tickets
5. Nice Fest: 3,048 tickets

EVENT_NAME ↑	TOTAL_TICKETS SOLD ↑	FIRST_SALE_DATE ↑	LAST_SALE_DATE ↑
Instead Fest	4514	20089	20252
Age Fest	3608	20089	20269
Least Fest	3287	20089	20258
Own Fest	3285	20089	20251

The data shows ticket sales spanning from January 1, 2025 to various dates in June 2025, with Instead Fest being the clear leader in ticket sales.

Fantastic, it gave us an answer with a chart of the top 5 events! I can open the SQL section and see the join in the SQL.

Show me the top 5 events by tickets sold.

#### Thinking steps

Hide Details ^

##### ✓ Planning the next steps

I'll help you find the top 5 events based on ticket sales. Let me query the music festival data using the cortex\_analyst\_text\_to\_sql tool.

##### ✓ Executing SQL ✨

```
WITH __events AS (
  SELECT
    event_name,
    event_id
  FROM si_events_hol.public.events
), __ticket_sales AS (
  SELECT
    event_id,
    ticket_id,
    sale_date
  FROM si_events_hol.public.ticket_sales
)
SELECT
  e.event_name,
  COUNT(ts.ticket_id) AS total_tickets_sold,
```

##### ✓ Planning the next steps

I have gathered sufficient data to answer the user's question. The SQL results from the previous tool call show a clear table with the top 5 events ranked by total tickets sold, including the event names and their corresponding ticket sales numbers. Since the user didn't specifically request a chart or any additional information, this data is complete and sufficient to answer their question.

We can ask another question: *What is the ratio of ticket types purchased for Instead Fest?*

What is the ratio of ticket types purchased for Instead Fest?

✓ Thinking completed ✅ 1 verified query

Show Details ▾

Based on the tool results from our previous conversation, here's the breakdown of ticket types for Instead Fest:

1. General Admission: 1,565 tickets (34.67% of total)
2. Early Bird: 1,492 tickets (33.05% of total)
3. VIP: 1,457 tickets (32.28% of total)

The distribution is remarkably even across all three ticket types, with each type representing roughly one-third of total ticket sales. General Admission has a slight lead with just over 34% of sales, followed by Early Bird tickets at 33%, and VIP tickets making up the remaining 32%.

TICKET_TYPE ↑	TYPE_COUNT ↑	PERCENTAGE ↑
General	1565	0.3466991582
Early Bird	1492	0.3305272486
VIP	1457	0.3227735933

A third question I could ask is: *Did any one customer go to multiple shows?* This would utilize a synonym we created (show) as well as utilize the relationship we created to the customers table:

Did any one customer go to multiple shows?

✓ Thinking completed ✅ 1 verified query

Show Details ▾

Yes, many customers attended multiple shows. Based on the tool results, there were quite a few customers who went to multiple events, with some particularly active attendees:

The most frequent attendees were:

- Michelle Pena and Eric Clayton: attended 12 different shows each
- Angela Douglas, Maria Foster, Thomas Valdez, Joseph Wheeler, and Ashley Ortiz: attended 11 shows each
- Multiple customers attended 10 shows, including Charlene Harper, Tina Walker, Kyle Smith, and others
- Several customers attended 9 shows

The data shows a significant number of repeat customers, with many people attending multiple festivals throughout the season. The distribution ranges from customers attending 9 shows up to 12 shows, demonstrating strong customer loyalty across multiple events.

CUSTOMER_NAME ↑	NUM_EVENTS ↑
Michelle Pena	12
Eric Clayton	12
Angela Douglas	11
Maria Foster	11

These answers appear to be correct. If they were incorrect, the first step would be to review the semantic model. This review would focus on verifying the accuracy of the defined relationships and identifying any potential synonyms that could enhance the model's intelligence and robustness.

## Step 5: Create Search Service

We have document data that has been parsed and loaded into a table named 'Contracts' from a 'Contracts.csv' file. To enable searching of this text data, we will create a search service. This can be done by navigating to AI & ML, then Cortex Search, and following the wizard.

**Snowflake AI & ML Studio**

**Fast, Easy and Secure AI and ML with Snowflake**

Snowflake offers two broad categories of powerful, intelligent features based on AI and Machine Learning (ML). These features can help you do more with your data in the cloud. Get started quickly with AI & ML in this Studio. [Learn more](#)

**Cortex Playground** PREVIEW

**LLM**

Generate a text response to your prompt using cutting-edge LLMs

**Cortex Search** PREVIEW

**LLM**

Create and manage search services over your data

**Cortex Search** PREVIEW

**Why use Cortex Search?**

Build performant, high-quality search bars and RAG chatbots on unstructured data.

Tutorials

**Cortex Search Documentation**

Learn more about creation, usage, and management of Cortex Search Services.

Documentation

**Create**

We can create it in our SNOWFLAKE\_INTELLIGENCE.CONFIG database and schema, and call it Festival\_Contract\_Search

## Create Search Service

New service

- Select data
- Select search column
- Select attributes
- Select columns
- Configure indexing



### Let's create a new Cortex Search Service

We'll guide you through the steps of selecting a data source, setting service parameters, and creating the Search Service.

Select a warehouse to power the service. This warehouse will be used for materializing the results of the source query upon creation and refresh.

#### Role and Warehouse

SNOWFLAKE\_INTELLIGENCE\_ADMI... • SNOWFLAKE\_INTELLIGENCE\_WH (...)

#### Database and Schema

SNOWFLAKE\_INTELLIGENCE.CONFIG ▾

#### Name

Festival\_Contract\_Search

Cancel

Next: Select data

Going to the next screen, it asks for the table that it wants to index. We will select a table that we uploaded earlier. This is in the SI\_EVENTS\_HOL.PUBLIC and our table is called CONTRACTS

## Select data to be indexed

Select a table or view containing containing text data that you'd like to search over. At least one column in this table or view must be a text column that you'd like to search over.

If you wish to specify multiple data sources or perform transformations when defining your service, please use the [SQL surface](#).

SI\_EVENTS\_HOL.PUBLIC ▾

Search Tables and Views

CONTRACTS  CUSTOMERS EVENTS TICKET\_SALES

The next screen asks us to choose which column we want to be able to search. In our case, the column name is TEXT, go ahead and select that column from the list

## Select a search column

The data in this column must be text-based and will be the column that your queries search over.

CONTRACTS

Search Column

DOCUMENT\_TITLE

URL

TEXT

Next, it is asking what attributes we want to bring in. We will choose both DOCUMENT\_TITLE and URL:

Optional

## Select attribute column(s)

Select a set of columns that you'd wish to use as filters when querying the service.

CONTRACTS

Q Search Column

Column	Selected
A DOCUMENT_TITLE	<input checked="" type="checkbox"/>
A URL	<input checked="" type="checkbox"/>
A TEXT	

We will leave all columns on the next page selected:

Optional

## Select columns to include in the service

Select the columns that you want to include in the search index. The data in these columns will be included in the search service and will be available for querying.

Search and attribute columns must be part of the included columns.

### CONTRACTS

Search Column

DOCUMENT\_TITLE

URL

TEXT

Select all

[Previous: Select attributes](#)

[Next: Configure indexing](#)

As this is a demonstration with unchanging data, the chosen lag time is inconsequential. Therefore, a 1-day lag can be applied.

## Configure your Search Service

Select your desired configuration parameters for the service.

Target Lag

1 days



Target Lag parameter specifies the maximum frequency with which the service will check for and materialize updates based on changes to its source data.

Embedding model (optional)

snowflake-arctic-embed-m-v1.5

Each model may incur a different cost per million input tokens processed. Refer to the [Snowflake Service Consumption Table](#).

For more information about each embedding model, refer to the [Cortex Search documentation](#).

Warehouse for indexing

SNOWFLAKE\_INTELLIGENCE\_WH

Indexing can take couple minutes to an hour depending on size of data. [See documentation](#).

Now our Search Service is created:

The screenshot shows the Cortex Search interface with the following details:

- Search Service:** FESTIVAL\_CONTRACT\_SEARCH (PREVIEW)
- Base table:** SNOWFLAKE\_INTELLIGENCE.CONFIG
- Table Name:** FESTIVAL\_CONTRACT\_SEARCH
- Columns:**

COLUMN	SEARCHABLE	ATTRIBUTE
TEXT	<input checked="" type="checkbox"/>	<input type="checkbox"/>
DOCUMENT_TITLE	<input type="checkbox"/>	<input checked="" type="checkbox"/>
URL	<input type="checkbox"/>	<input checked="" type="checkbox"/>
- Rows:** 0 rows

## Step 10: Add to Agent

We will reuse the previously created "Music Festival Agent" and integrate the Search Service we built into it. To do this, select the existing agent and then proceed to add the Search Service.

We will then add the search service by selecting where we created it which was in SNOWFLAKE\_INTELLIGENCE.CONFIG we will call it Festival Contract Documents and make sure we define the URL column in our search service which is called URL:

## Step 10: Test with Updated Agent

Let's go back to Snowflake Intelligence and make sure our new search service is available to query, and then

Now that we see our Snowflake Data there, we know it will utilize our search service we created. Let's ask it a question:

*What is the cancellation policy for the band Algorhythms?*

Notice that no SQL is generated or written, as it is using our search service. It gives us the answer and the source of the document from which this information came.

What is the cancellation policy for the band Algorhythms?

✓ Thinking completed Show Details ▾

1 🎶 The Algorhythms wil... 2 🎶 The Algorhythms wil... 3 🎶 Sponsor agrees to p... Show More

Here's the cancellation policy for the Algorhythms:

If the Festival cancels:

- 25% of the guarantee fee (\$100,000) will be paid if canceled between 60 and 30 days before the performance ①
- 50% of the guarantee fee will be paid if canceled less than 30 days before the performance ①

If the Algorhythms (Artist) cancels:

- No fee is owed by the Promoter ①