

Getting Started with Unstructured Data

Overview

This Quickstart is designed to help you understand the capabilities included in Snowflake's support for unstructured data. Sign up for a free 30-day trial of Snowflake and follow along with this lab exercise. After completing this lab, you'll be ready to start storing and managing your own unstructured data in Snowflake.

Prerequisites

- Snowflake account
- Basic knowledge of SQL, database concepts, and objects

What You'll Learn

- How to access and store unstructured data
- How to govern unstructured data
- How to search for unstructured data using directory tables
- How to securely share unstructured data
- How to process unstructured data

What You'll Build

- A stage for storing and accessing files in Snowflake
- A user-defined function using Snowflake's engine to process files
- A secure view to share in the Snowflake Marketplace

Notice and Terms of Use

The data provided for this lab is an extract from the Enron email database made available by Carnegie Mellon University (<https://www.cs.cmu.edu/~enron/>).

Use of the data provided is limited to this quickstart in connection with the Snowflake service and is subject to any additional terms and conditions on the Carnegie Mellon site.

By accessing this data, you acknowledge and agree to the limits and terms related to the use of the Enron email data.

Prepare your lab environment

If you haven't already, register for a [Snowflake free 30-day trial](#). The Snowflake edition (Standard, Enterprise, Business Critical, e.g.), cloud provider (AWS, Azure, e.g.), and Region (US East, EU, e.g.) do not matter for this lab. We suggest you select the region which is physically closest to you and the Enterprise Edition, our most popular offering. After registering, you will receive an email with an activation link and your Snowflake account URL.

Navigating to Snowsight

For this lab, you will use the latest Snowflake web interface, Snowsight.

1. Log into your Snowflake trial account
2. Click on **Snowsight** Worksheets tab. The new web interface opens in a separate tab or window.
3. Click **Worksheets** in the left-hand navigation bar. The **Ready to Start Using Worksheets and Dashboards** dialog opens.
4. Click the **Enable Worksheets and Dashboards** button.



Ready to start using Worksheets and Dashboards?

Snowflake's next-generation Worksheets and Dashboards are ready to be enabled for your account.

Whenever you're ready, click the button below to enable Worksheets and Dashboards for all users in your account. Worksheets will still be available in the Classic UI.

[Enable Worksheets and Dashboards](#)

Store & Access Unstructured Data

Let's start by preparing to load the unstructured data into Snowflake. Snowflake supports two types of stages for storing data files used for loading and unloading:

- [Internal stages] store the files internally within Snowflake.
- [External stages] store the files in an external location (i.e. S3 bucket) that is referenced by the stage. An external stage specifies location and credential information, if required, for the bucket.

Create a Database, Schema, and Warehouse

Before creating any stages, let's create a database and a schema that will be used for loading the unstructured data. We will use the UI within the Worksheets tab to run the DDL that creates the database and schema. Copy the commands below into your trial environment, and execute each individually.

```
use role sysadmin;

create or replace database emaildb comment = 'Enron Email Corpus Database';
create or replace schema raw;
create or replace warehouse quickstart;

use database emaildb;
use schema raw;
use warehouse quickstart;
```

Access Unstructured Data Stored in an S3 Bucket

The data we will be using in this lab is stored in an S3 bucket.

Create an External Stage

You are working with unstructured PDFs that have already been staged in a public, external S3 bucket. Before you can use this data, you first need to create a Stage that specifies the location of our external bucket.

For this lab we are using an AWS S3 bucket in us-east-1. To minimize data egress/transfer costs in the future, you should select a staging location from the same cloud provider and region as your Snowflake environment.

Grant the `PUBLIC` schema access to the database, schema, and warehouse just created (This will be relevant in the next section).

```
use role sysadmin;
grant usage on database emaildb to public;
grant usage on schema emaildb.raw to public;
grant usage on warehouse quickstart to public;
```

From the same worksheet you've been using, run this command to create an external stage called `email_stage`.

```
use schema emaildb.raw;
-- Create an external stage where files are stored.
create or replace stage email_stage
url = "s3://sfquickstarts/Getting Started Unstructured Data/Emails/mailbox/"
directory = (enable = true auto_refresh = true);
```

You can run this command to see a list of the files in your external stage.

```
ls @email_stage;
```

22

23

ls @email_stage;

Objects

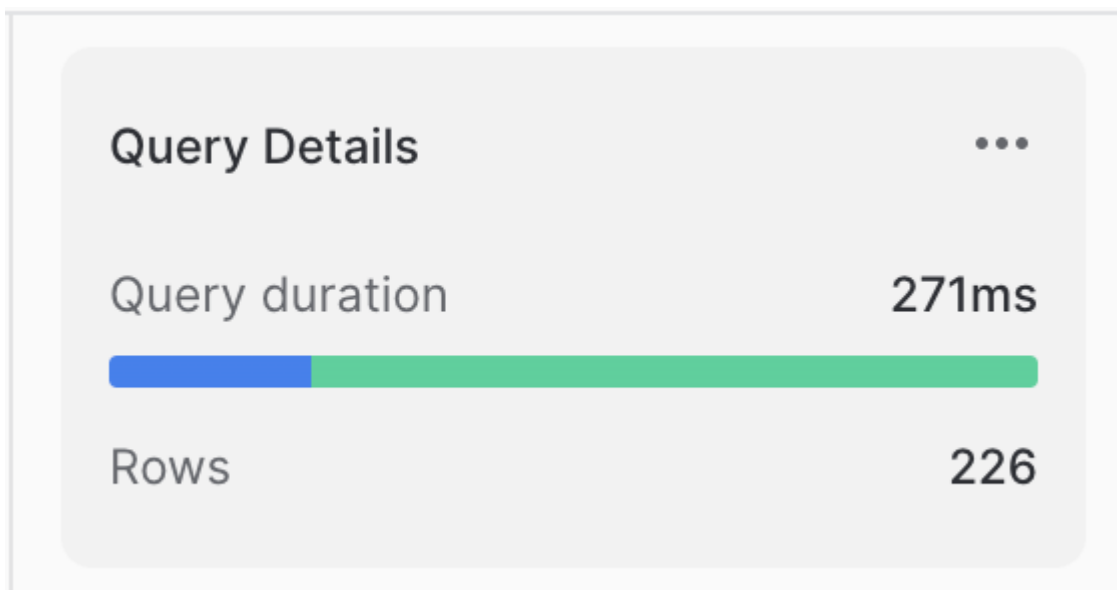
Query

Results

Chart

	name	...	size
1	s3://sfquickstarts/Getting Started Unstructured Data/Emails/mailbox/arnold-j/inbox,		886
2	s3://sfquickstarts/Getting Started Unstructured Data/Emails/mailbox/arnold-j/inbox,		886
3	s3://sfquickstarts/Getting Started Unstructured Data/Emails/mailbox/arnold-j/inbox,		917
4	s3://sfquickstarts/Getting Started Unstructured Data/Emails/mailbox/arnold-j/inbox,		1,010

We can see that the stage contains various mailboxes from Enron users containing some files (which are text files) of various sizes. We can get a summary of the file corpus in the query results on the right-hand side. For the purpose of this quickstart, just a sample of 226 files out of 517,551 files has been extracted.



The size of the files: We can see that the file size ranges from 666 bytes to 1023 bytes, with the majority of the files closer to 1023 bytes. If we click on the size metric, we can get more detailed information. The total corpus size is 208,414 bytes, with an average size of 922 bytes. If we hover over the histogram, we can filter results based on file size.

Store Unstructured Data in an Internal Stage

Alternatively, you can store data directly in Snowflake with internal stages. Now, we want to create an internal stage and upload the same files while maintaining the directory structure of the various individual mailboxes on an internal stage.

Create an Internal Stage

Run this command to create an internal stage called `email_stage_internal` as follows.

```
use schema emaildb.raw;
create or replace stage email_stage_internal
directory = (enable = TRUE)
encryption = (type = 'SNOWFLAKE_SSE');
```

Note that we have to specify a server-side encryption on the internal stage. When using the default client-side encryption, the files will be returned encrypted and not readable when accessed through URLs (in Section 6).

Download Data and Scripts

We need to first download the following files to the local workstation by clicking on the hyperlinks below. The subsequent steps require [SnowSQL CLI] installed on the local workstation where the lab is ran:

- [upload.snf](#): SnowSQL Script which will upload the files to the internal stage just created.
- [mailbox.tar.gz](#): The actual email data.

Once downloaded, untar the contents of the files on your local workstation and note the full path including the mailbox parent directory of the tar archive. In the example below, this path is

`/Users/znh/Downloads/quickstart/`. The file can be untarred as follows.

```
cd /Users/znh/Downloads/quickstart/  
tar xzvf mailbox.tar.gz
```

Upload Files using SnowSQL

Before opening terminal, find out your account identifier which for the trial account will be `<account-locator>`. `<region-id>`.`<cloud>` . These fields can be retrieved from the URL of your Snowflake account.

For example, the URL to access the trial account is `https://xx74264.ca-central-1.aws.snowflakecomputing.com/` . These are the values for the account identifier:

- Account Locator: `xx74264`
- Region ID: `ca-central-1`
- Cloud: `aws`

There may be additional segments if you are using your own account part of an organization. You can find those from the URL of your Snowflake account. Please check the [Snowflake Documentation] for additional details on this topic.

Now open a terminal on your workstation and run the following SnowSQL command. You will be prompted for the password for the Snowflake user passed as a parameter.

```
snowsql -a <account-identifier> \  
-u <user-id> -d emaildb -r sysadmin -s raw \  
-D srcpath=<source-path> \  
-D stagename=@email_stage_internal \  
-o variable_substitution=true -f upload.snf
```

Using the examples above for the path and the account identifier, and the userid `myuser` , the command would be:

```
snowsql -a xx74264.ca-central-1.aws \  
-u myuser -d emaildb -r sysadmin -s raw \  
-D srcpath=/Users/znh/Downloads/quickstart/mailbox \  
-D stagename=@email_stage_internal \  
-o variable_substitution=true -f /Users/znh/Downloads/quickstart/upload.snf
```

The upload may take a few seconds depending on the speed of your internet connection. If the upload is successful, you should see data being uploaded in each subfolder, and you may see an output like the following on your terminal.

172.	172.	1023	1023	NONE	NONE	UPLOADED	
200.	200.	972	972	NONE	NONE	UPLOADED	
27.	27.	971	971	NONE	NONE	UPLOADED	
36.	36.	964	964	NONE	NONE	UPLOADED	
5.	5.	1002	1002	NONE	NONE	UPLOADED	
60.	60.	976	976	NONE	NONE	UPLOADED	
83.	83.	879	879	NONE	NONE	UPLOADED	
98.	98.	802	802	NONE	NONE	UPLOADED	
14 Row(s) produced. Time Elapsed: 0.978s							
source	target	source_size	target_size	source_compression	target_compression	status	message
53.	53.	1018	1018	NONE	NONE	UPLOADED	
6.	6.	933	933	NONE	NONE	UPLOADED	
813.	813.	1010	1010	NONE	NONE	UPLOADED	
3 Row(s) produced. Time Elapsed: 0.623s							
source	target	source_size	target_size	source_compression	target_compression	status	message
121.	121.	952	952	NONE	NONE	UPLOADED	
131.	131.	800	800	NONE	NONE	UPLOADED	
272.	272.	998	998	NONE	NONE	UPLOADED	
38.	38.	974	974	NONE	NONE	UPLOADED	
39.	39.	882	882	NONE	NONE	UPLOADED	
53.	53.	912	912	NONE	NONE	UPLOADED	
9.	9.	964	964	NONE	NONE	UPLOADED	
7 Row(s) produced. Time Elapsed: 0.840s							
source	target	source_size	target_size	source_compression	target_compression	status	message
233.	233.	948	948	NONE	NONE	UPLOADED	
1 Row(s) produced. Time Elapsed: 0.698s							
source	target	source_size	target_size	source_compression	target_compression	status	message
196.	196.	756	756	NONE	NONE	UPLOADED	
540.	540.	884	884	NONE	NONE	UPLOADED	
2 Row(s) produced. Time Elapsed: 0.624s							
source	target	source_size	target_size	source_compression	target_compression	status	message
103.	103.	855	855	NONE	NONE	UPLOADED	
1 Row(s) produced. Time Elapsed: 0.538s							

Verify if the files have been uploaded successfully by entering the following command on your Snowflake worksheet.

```
ls @email_stage_internal;
```

You should now see an identical list of files uploaded to the internal stage. Make sure you see 226 files uploaded

62

ls @email_stage_internal;

	name	size	md5	last_modified
1	email_stage_internal/arnold-j/inbox/159.	886	c7dd3c166fb2bb50651c423b42afd9c8	Thu, 3 Feb 2022 06:35:07 GMT
2	email_stage_internal/arnold-j/inbox/167.	886	2360c3e18e4b56b835b9e81960ee9b60	Thu, 3 Feb 2022 06:35:08 GMT
3	email_stage_internal/arnold-j/inbox/4.	917	7896386dfec070d231ac6cbc5e22c9f	Thu, 3 Feb 2022 06:35:08 GMT
4	email_stage_internal/arnold-j/inbox/94.	1,010	b92a3f462a2be095195d53a4feb07bf7	Thu, 3 Feb 2022 06:35:08 GMT
5	email_stage_internal/arora-h/inbox/2.	950	6b46b9a7766fd2bf6a361c44c7b2d9d2	Thu, 3 Feb 2022 06:35:08 GMT
6	email_stage_internal/bass-e/inbox/137.	776	05e89dc33d1ed12309797d8d75061c5c	Thu, 3 Feb 2022 06:35:09 GMT
7	email_stage_internal/bass-e/inbox/180.	1,016	271c5658aa58f11dbdd40628802e63e5	Thu, 3 Feb 2022 06:35:09 GMT
8	email_stage_internal/bass-e/inbox/28.	881	274fbfae0dfe155ee42f12c3365dcaf7	Thu, 3 Feb 2022 06:35:09 GMT
9	email_stage_internal/beck-s/inbox/27.	856	bd914e5abef09146f60f14c6399b0663	Thu, 3 Feb 2022 06:35:10 GMT
10	email_stage_internal/beck-s/inbox/307.	1,010	c9dda2c460ee2807a6788db6dc963eec	Thu, 3 Feb 2022 06:35:10 GMT

Query Details

Query duration

103ms

Rows

226

name

Aa

100% filled

size

123

666

1,023

Govern Unstructured Data Access

Just like structured and semi-structured data, access permissions to unstructured data in Snowflake can be governed using role-based access control (RBAC).

Create Role and Grant Access

Let's create a role that will provide read access to the external stage we've already created that contains PDFs.

Let's first create the analyst role.

```
use role accountadmin;
create or replace role analyst;
grant role analyst to role sysadmin;
```

Then, switch back to `sysadmin` role, and grant the role `analyst` the rights to use the database `emaildb`, and the schema `raw` we just created earlier, as well as the ability to `read` from the stage.

```
grant usage on database emaildb to role analyst;
grant usage on schema emaildb.raw to role analyst;
grant usage on warehouse quickstart to role analyst;
grant read on stage email_stage_internal to role analyst;
```

You can verify the `analyst` role only has access to read by listing the files in the internal stage, then trying to remove files from the external stage. When trying to remove, this should result in an error message.

```
use role analyst;

-- List files from the stage. This should execute successfully.
ls @email_stage_internal;

-- Try to remove files from the stage. This should return an error.
rm @email_stage_internal;
```

```
74 use role analyst;
75
76 -- List files from the stage. This should execute successfully.
77 ls @email_stage_internal;
78
79 -- Try to remove files from the stage. This should return an error.
80 rm @email_stage_internal;
```

Objects Query Results Chart



SQL access control error:

Insufficient privileges to operate on stage 'EMAIL_STAGE_INTERNAL'

In the subsequent sections, we will see a more fine-grained access control of the different unstructured files stored in Snowflake using scoped URLs.

Catalog Unstructured Data using Directory Tables

One of the main pain points in managing large repositories of unstructured data is the ability to access metadata easily on the numerous files, as well as retrieve files per some metadata attributes (last modified, file size, file patterns).

Directory Tables are built-in tables in Snowflake that provide an up-to-date, tabular file catalog for external and internal stages. Directory Tables make it easy to search for and query files using SQL.

We will first reset the session parameters to the correct role, virtual warehouse, database and schema:

```
use role sysadmin;
use warehouse quickstart;
use schema emaildb.raw;
```

Prior to accessing the directory, it needs to be refreshed first for the files previously uploaded using the following command.

```
alter stage email_stage_internal refresh;
```

Run the following command to access the directory table.

```
select *
from directory(@email_stage_internal);
```

This will provide some detailed metadata information about the files stored in the stage including the `RELATIVE_PATH`, the `LAST_MODIFIED` timestamp, the `SIZE`, the `ETAG` as well as the `FILE URL` (more information on this in the next section).

89
90
91

```
select *
from directory(@email_stage_internal);
```

Objects Query Results Chart

	RELATIVE_PATH	SIZE	LAST_MODIFIED	MD5	ETAG
1	arnold-j/inbox/159.	886	2022-02-02 22:35:07.000 -0800	c7dd3c166fb2bb50651c423b42afd9c8	c7dd3c166fb2bb50651c423b42afd9c8
2	arnold-j/inbox/167.	886	2022-02-02 22:35:08.000 -0800	2360c3e18e4b56b835b9e81960ee9b60	2360c3e18e4b56b835b9e81960ee9b60
3	arnold-j/inbox/4.	917	2022-02-02 22:35:08.000 -0800	7896386dfeec070d231ac6cbc5e22c9f	7896386dfeec070d231ac6cbc5e22c9f
4	arnold-j/inbox/94.	1,010	2022-02-02 22:35:08.000 -0800	b92a3f462a2be095195d53a4feb07bf7	b92a3f462a2be095195d53a4feb07bf7
5	arora-h/inbox/2.	950	2022-02-02 22:35:08.000 -0800	6b46b9a7766fd2bf6a361c44c7b2d9d2	6b46b9a7766fd2bf6a361c44c7b2d9d2
6	bass-e/inbox/137.	776	2022-02-02 22:35:09.000 -0800	05e89dc33d1ed12309797d8d75061c5c	05e89dc33d1ed12309797d8d75061c5c

Searching Directory Tables

We could now query these files using some SQL commands. For example, let's assume we want to identify all the emails from the mailbox belonging to the user `nemec-g`. We could easily do this with the following SQL query.

```
select *
from directory(@email_stage_internal)
where RELATIVE_PATH like '%nemec-g%';
```



```

92 |
93 | select *
94 | from directory(@email_stage_internal)
95 | where RELATIVE_PATH like '%nemec-g%';

```

	RELATIVE_PATH	SIZE	LAST_MODIFIED	MD5	ETAG	FILE
1	nemec-g/inbox/106.	995	2022-02-02 22:35:37.000 -0800	e1395972e03e89741f1df457151b0c2e	e1395972e03e89741f1df457151b0c2e	http
2	nemec-g/inbox/1138.	856	2022-02-02 22:35:35.000 -0800	9bd33a76387d83f58a9cf0916aba4a01	9bd33a76387d83f58a9cf0916aba4a01	http
3	nemec-g/inbox/1222.	1,008	2022-02-02 22:35:35.000 -0800	8c3cf972b4ef94fe20f5dcbf436d3c96	8c3cf972b4ef94fe20f5dcbf436d3c96	http
4	nemec-g/inbox/1225.	909	2022-02-02 22:35:35.000 -0800	8ed2ca537b3171fbd0c45a7302109e1	8ed2ca537b3171fbd0c45a7302109e1	http
5	nemec-g/inbox/1263.	986	2022-02-02 22:35:36.000 -0800	ec58b5156d78d98da4baf24584503e0a	ec58b5156d78d98da4baf24584503e0a	http
6	nemec-g/inbox/1316.	952	2022-02-02 22:35:37.000 -0800	577548b04b32916ec62fb53954858451	577548b04b32916ec62fb53954858451	http
7	nemec-g/inbox/141.	919	2022-02-02 22:35:37.000 -0800	22cedf8a1d227002c22ae4ce8d0b80e1	22cedf8a1d227002c22ae4ce8d0b80e1	http
8	nemec-g/inbox/303.	997	2022-02-02 22:35:36.000 -0800	55b9e5a68be1fe758eea1b56b510958a	55b9e5a68be1fe758eea1b56b510958a	http

This query returns the 13 emails belonging to that user. Now let's try to identify the 5 largest email text in the dataset. We can do that with the following query.

```

select *
from directory(@email_stage_internal)
order by size desc
limit 5;

```

```

96 |
97 | select *
98 | from directory(@email_stage_internal)
99 | order by size desc
100 | limit 5;

```

	RELATIVE_PATH	SIZE	LAST_MODIFIED	MD5	ETAG	FILE
1	white-s/inbox/172.	1,023	2022-02-02 22:35:54.000 -0800	071f9b499fd89f8ea9121d37255105c9	071f9b499fd89f8ea9121d37255105c9	https
2	crandell-s/inbox/198.	1,022	2022-02-02 22:35:15.000 -0800	5c0f9e9e02e84711820a2cdd32897220	5c0f9e9e02e84711820a2cdd32897220	https
3	mckay-b/inbox/24.	1,022	2022-02-02 22:35:34.000 -0800	a84e395e6bc0ea689bf969c9d176b349	a84e395e6bc0ea689bf969c9d176b349	https
4	lay-k/inbox/552.	1,021	2022-02-02 22:35:30.000 -0800	a7a3e933d1d472341a188516142cfe6b	a7a3e933d1d472341a188516142cfe6b	https
5	reitmeyer-j/inbox/51.	1,020	2022-02-02 22:35:40.000 -0800	83ff08f2477116c695826a16733bfcd6	83ff08f2477116c695826a16733bfcd6	https

Automatic Refresh

Say you want the directory table to refresh whenever a file is added to your S3 bucket. This can be accomplished by using event notifications in S3. When a new file is added to a bucket, S3 will send a notification to Snowflake, and a Stream can refresh the directory table.

In this quickstart, we won't setup notifications in S3, but the command below is what you would use to create a stream on the directory table for a stage. More detailed documentation for automatically refreshing directory tables can be found [\[here\]](#).

```

-- Create a table stream on directory table
create stream documents_stream on directory(<stage_name>);

```

URLs for Secure Access

In the previous sections, we have seen how to store unstructured data in Snowflake, as well as access metadata about the unstructured files, and build queries to retrieve files based on metadata filters.

In this section, we will look into how Snowflake offers access to the unstructured data through various types of URLs, as well as provide a more granular governance over unstructured data than at the stage level, as reviewed previously

in Section 2.

There are three different types of URLs that you can use to access unstructured data:

- **[Scoped URL]:** A scoped file URL can be generated for a user to give the user short-lived, scoped access to the file without giving privileges on the stage.
- **[File URL]:** A file URL requires a user to be authenticated with Snowflake and requires the user to have read privileges on the stage.
- **[Pre-signed URL]:** As the name suggests, pre-signed URLs are already authenticated. Users can simply download the files using pre-signed URLs.

The URL format for files is `https://.snowflakecomputing.com/api/files/`.

Scoped URL

Scoped URLs are encoded URLs that permit temporary access to a staged file without granting privileges to the stage. The URL expires when the persisted query result period ends (i.e. the results cache expires), which is currently 24 hours.

We can generate a scoped URL of any file using the function `build_scoped_url()`. For example, let's create a view which provides the scoped URL for files in the stage `email_stage_internal`.

```
create or replace view email_scoped_url_v
as
select
    relative_path
    , build_scoped_file_url(@email_stage_internal,relative_path) as scoped_url
from directory(@email_stage_internal);

select * from email_scoped_url_v limit 5;
```

108
109 | select * from email_scoped_url_v limit 5;

	RELATIVE_PATH	SCOPED_URL
1	arnold-j/inbox/159.	https://uaa85655.snowflakecomputing.com/api/files/01a2aec7-0501-3dc9-003d-7683000b902e/17300278592315398/e7P64dySMcqv3ZEswLkbJJF7V
2	arnold-j/inbox/167.	https://uaa85655.snowflakecomputing.com/api/files/01a2aec7-0501-3dc9-003d-7683000b902e/17300278592315398/ETahC59q5NAkfhd6NBKxJqTCV
3	arnold-j/inbox/4.	https://uaa85655.snowflakecomputing.com/api/files/01a2aec7-0501-3dc9-003d-7683000b902e/17300278592315398/vOhC8sAVIC8D%2b0fep9lq2Hh
4	arnold-j/inbox/94.	https://uaa85655.snowflakecomputing.com/api/files/01a2aec7-0501-3dc9-003d-7683000b902e/17300278592315398/q5yqO4P8jSjVphS%2foHNGkNc
5	arora-h/inbox/2.	https://uaa85655.snowflakecomputing.com/api/files/01a2aec7-0501-3dc9-003d-7683000b902e/17300278592315398/KRF%2bD4eZGTxg3m06zIMS

As explained previously, this URL will be valid for 24 hours. Snowsight retrieves the file only for the user who generated the scoped URL.

Scoped URLs enable access to the files via a view that retrieves scoped URLs. Only roles that have privileges on the view can access the files. The scoped URL contents are all encrypted and doesn't give any information about the bucket, database or schema.

Secure Access with RBAC and Scoped URL

Let's assume a scenario where the `analyst` role needs access to all the files from the inbox of `NEMEC-G`. We can build a dynamic view which will filter the output based on the role of the view user.

First let's build an assignment table where various roles (or even users) can be assigned a given mailbox for further analysis.

```
create or replace table assignment (mailbox string, role string, filter string);

insert into assignment values ('NEMEC-G', 'ANALYST', '%nemec-g%');
insert into assignment values ('*', 'SYSADMIN', '%');

select * from assignment;
```

As we can see from the query output above, the assignment table controls a 1-to-1 role-to-mailbox access mapping.

We can now build a SQL view which will join with the assignment table to dynamically filter rows based on the role of the user executing the view and grant access to the view to the `analyst` role:

```
create or replace secure view analyst_file_access_v as
select
    relative_path
    , build_scoped_file_url(@email_stage_internal, relative_path) as scoped_url
from directory(@email_stage_internal)
inner join assignment
where relative_path like filter
    and role = current_role();

grant select on analyst_file_access_v to role analyst;
```

Let's switch to role `analyst` and query the view.

```
use role analyst;
use warehouse quickstart;
use schema raw;
select * from analyst_file_access_v;
```

```
117 use role analyst;
118 use warehouse quickstart;
119 select * from analyst_file_access_v;
```

	RELATIVE_PATH	SCOPED_URL
1	nemec-g/inbox/106.	https://uua85655.snowflakecomputing.com/api/files/01a2294e-0401-272c-003d-768300071042/1730027859231
2	nemec-g/inbox/1138.	https://uua85655.snowflakecomputing.com/api/files/01a2294e-0401-272c-003d-768300071042/1730027859231
3	nemec-g/inbox/1222.	https://uua85655.snowflakecomputing.com/api/files/01a2294e-0401-272c-003d-768300071042/1730027859231
4	nemec-g/inbox/1225.	https://uua85655.snowflakecomputing.com/api/files/01a2294e-0401-272c-003d-768300071042/1730027859231
5	nemec-g/inbox/1263.	https://uua85655.snowflakecomputing.com/api/files/01a2294e-0401-272c-003d-768300071042/1730027859231
6	nemec-g/inbox/1316.	https://uua85655.snowflakecomputing.com/api/files/01a2294e-0401-272c-003d-768300071042/1730027859231
7	nemec-g/inbox/141.	https://uua85655.snowflakecomputing.com/api/files/01a2294e-0401-272c-003d-768300071042/1730027859231

Click the `scoped_url` corresponding to the file `1222.` to your workstation and review the file locally.

If we switch the role to `sysadmin` and run the same query:

```
use role sysadmin;
use warehouse quickstart;
use schema raw;
select * from analyst_file_access_v;
```

This returns scoped URLs to access all the files.

As we just experimented, scoped URLs are ideal for use in custom applications, providing unstructured data to other accounts via a share, or for downloading and ad-hoc analysis of unstructured data via Snowsight.

File URL

A file URL is a permanent URL that identifies the database, schema, stage, and file path to a set of files, as opposed to the previous scoped URL where all this information is encrypted. A role that has sufficient privileges on the stage can access the files. It does not contain any authentication token. Authentication needs to be done when connecting through REST API. However, it works from our current authenticated Snowsight session.

The following query will provide the file urls for the mailbox `arnold-j`.

```
select
  relative_path
  , build_stage_file_url(@email_stage_internal,relative_path) as stage_file_url
from directory(@email_stage_internal)
where relative_path like '%arnold-j%';
```

```
126
127 | select
128 |     relative_path
129 |     , build_stage_file_url(@email_stage_internal,relative_path) as stage_file_url
130 | from directory(@email_stage_internal)
131 | where relative_path like '%arnold-j%';
```

	RELATIVE_PATH	STAGE_FILE_URL
1	arnold-j/inbox/159.	https://uua85655.snowflakecomputing.com/api/files/EMAILDB/RAW/EMAIL_STAGE_INTERNAL/arnold-j%2finbox%2f159%2e
2	arnold-j/inbox/167.	https://uua85655.snowflakecomputing.com/api/files/EMAILDB/RAW/EMAIL_STAGE_INTERNAL/arnold-j%2finbox%2f167%2e
3	arnold-j/inbox/4.	https://uua85655.snowflakecomputing.com/api/files/EMAILDB/RAW/EMAIL_STAGE_INTERNAL/arnold-j%2finbox%2f4%2e
4	arnold-j/inbox/94.	https://uua85655.snowflakecomputing.com/api/files/EMAILDB/RAW/EMAIL_STAGE_INTERNAL/arnold-j%2finbox%2f94%2e

Click on any of the stage file URLs and download the file to your workstation.

Pre-signed URL

Pre-signed URLs are used to download or access files, via a web browser for example, without authenticating into Snowflake or passing an authorization token. These URLs are ideal for business intelligence applications or reporting tools that need to display the unstructured file contents.

Pre-signed URLs are open but temporary. The expiration time for the access token is configurable when generating the URL. Any user or application can directly access or download the files until the expiration time is reached.

The following query will generate the pre-signed URLs for the mailbox `beck-s`. In the `get_presigned_url()` system function, we pass the parameter `300` which represents the expiration time for the token in seconds (which is 300s, so 5 minutes in this case).

```
select
  relative_path
  , get_presigned_url(@email_stage_internal,relative_path,300) as presigned_url
from directory(@email_stage_internal)
where relative_path like '%beck-s%';
```

Click on any cell value in the `PRESIGNED_URL` column. This should give the actual full URL on the right hand side in the grey cell. (Click to Copy)

```

132
133
134 select
135     relative_path
136     , get_presigned_url(@email_stage_internal,relative_path,300) as presigned_url
137 from directory(@email_stage_internal)
where relative_path like '%beck-s%';

```

	RELATIVE_PATH	PRESIGNED_URL
1	beck-s/inbox/27.	https://sfc-prod2-ds1-20-customer-stage.s3.us-west-2.amazonaws.com/ihuu-s-p2sv2850/stages/211abb7d-241e-4a48-ba1a-d4f1c
2	beck-s/inbox/307.	https://sfc-prod2-ds1-20-customer-stage.s3.us-west-2.amazonaws.com/ihuu-s-p2sv2850/stages/211abb7d-241e-4a48-ba1a-d4f1c
3	beck-s/inbox/413.	https://sfc-prod2-ds1-20-customer-stage.s3.us-west-2.amazonaws.com/ihuu-s-p2sv2850/stages/211abb7d-241e-4a48-ba1a-d4f1c
4	beck-s/inbox/537.	https://sfc-prod2-ds1-20-customer-stage.s3.us-west-2.amazonaws.com/ihuu-s-p2sv2850/stages/211abb7d-241e-4a48-ba1a-d4f1c
5	beck-s/inbox/638.	https://sfc-prod2-ds1-20-customer-stage.s3.us-west-2.amazonaws.com/ihuu-s-p2sv2850/stages/211abb7d-241e-4a48-ba1a-d4f1c

Open a new tab in your web browser, and paste the copied URL. This should download the file on your workstation. As you can see, the URL is valid even when executed outside from the Snowsight UI.

Perform Natural Language Processing

We have so far reviewed how to store unstructured data files, retrieve them, provide granular access to the files through various URLs and through secure views. In this section, we want to extract additional attributes from the files.

The entities extracted are going to be person names mentioned in the emails, as well as locations. The goal is to have these additional attributes used to enrich the file-level metadata for analytics.

The Java Code

The Java for the user-defined function (UDF) has already been written and provided below. This code uses the open source [Apache OpenNLP](#) library to perform natural language processing on English text in this occurrence.

The Java code leverages pre-built [machine learning models](#) to perform named entity extraction for persons and locations. These models are packaged manually post-build in a Fat JAR.

At a high level, the code does the following:

- Parse the text file contents using an Apache Tika text parser.
- Tokenize the parsed contents using a model.
- From the tokens, perform a named entity extraction for persons and locations using pre-built ML models.
- Serialize the results in a JSON string returned as an output.

```

public String ParseText(String filePath) throws IOException, SAXException,
TikaException {

    // Configure gson
    GsonBuilder gsonBuilder = new GsonBuilder()
    Gson gson = gsonBuilder.create();

    //detecting the file type
    BodyContentHandler handler = new BodyContentHandler();
    Metadata metadata = new Metadata();
    ParseContext pcontext=new ParseContext();

    SnowflakeFile file = SnowflakeFile.newInstance(filePath);

```

```

InputStream ins = file.getInputStream();

//Text document parser
TXTParser TextParser = new TXTParser();
TextParser.parse(ins, handler, metadata, pcontext);

String Contents = handler.toString();

String[] AllPersonEntities = null;
String[] AllLocationEntities = null;
String JsonResult = null;

NamedEntities NE = new NamedEntities(AllPersonEntities, AllLocationEntities);

try {
    for(int i=0;i<sentences.length;i++){
        String Tokens[] = new
NamedEntityExtraction().ParseTokens(Contents);
        String PersonEntities[] = new
NamedEntityExtraction().findName(Tokens);
        String LocationEntities[] = new
NamedEntityExtraction().findLocation(Tokens);

        AllPersonEntities = ArrayUtils.addAll(AllPersonEntities,
PersonEntities);
        AllLocationEntities = ArrayUtils.addAll(AllLocationEntities,
LocationEntities);

        NE.setPersons(AllPersonEntities);
        NE.setLocations(AllLocationEntities);

        JsonResult = gson.toJson(NE).toString();

    } catch (IOException e) {
        e.printStackTrace();
    }
    return(JsonResult);
}

```

A few elements relevant for this code:

- Notice the main class is **NamedEntityExtraction**.
- The method **ParseText** will be invoked by the UDF in the next section.
- The file path is passed as a parameter. It can be a URL to the file, or the path on the stage.

Creating a UDF in Snowflake

The precompiled jar file including all the dependencies has been uploaded and available on a Snowflake s3 public bucket. Creating the UDF involves a few steps in Snowflake.

1. Create the external stage mapping to the S3 bucket URI where the jar file is currently available. From the Snowflake worksheet, enter the following command:

```

use role sysadmin;
use schema emaildb.raw;

create or replace stage jars_stage_external
url = "s3://sfquickstarts/Common JARs/"
directory = (enable = true auto_refresh = true);

```

From the Snowflake worksheet, you can run the following command to confirm the jar file is listed in the external stage.

```
ls @jars_stage_external;
```

```

131
132 | ls @jars_stage_external;

```

Objects Query Results Chart

	name	size
1	s3://sfquickstarts/Common JARs/EmailNLV3-3.0.jar	141,144,554
2	s3://sfquickstarts/Common JARs/dcm4che-core-5.24.2.jar	506,805
3	s3://sfquickstarts/Common JARs/gson-2.8.7.jar	240,400

2. We can now create the UDF in Snowflake as the following.

```

create or replace function parseText(file string)
returns string
language java
imports = ('@jars_stage_external/EmailNLV3-3.0.jar')
handler = 'NamedEntityExtraction.ParseText'
;

```

Invoking the Java UDF

The UDF can be invoked on any text file containing readable english. From the email corpus stored on the internal stage, we can invoke the UDF as follows.

```

select parseText('@email_stage_internal/sanders-r/inbox/60.')
as entities_extraction;

```

```

147
148 | select parseText('@email_stage_internal/sanders-r/inbox/60.')
149 | as entities_extraction;

```

Objects Query Results Chart

	ENTITIES_EXTRACTION	As ENTITIES_EXTRACTION
1	{"Persons":["David","Richard","Rob","Mark","Richard","Vince Carter"],"Locations":["Toronto","Toronto"]}	{ "Persons": ["David", "Richard", "Rob", "Mark", "Richard", "Vince Carter"], "Locations": ["Toronto", "Toronto"] }

The output is actually serialized as a valid JSON format by the UDF. It contains 2 arrays, one for each named entities extraction:

```
{
  "Persons": ["David", "Richard", "Rob", "Mark", "Richard", "Vince Carter"],
  "Locations": ["Toronto", "Toronto"]
}
```

Since we are using pre-built models which haven't been trained on this particular corpus, the entity extraction may not always be accurate. However, the models perform overall quite well for illustration purposes for this quickstart.

Extracting and Storing Named Entities

We want to store the named entities as additional attributes for analysts to be able to select and retrieve the files of interest in their analysis, as well as perform some analytics on the attributes found.

We first want to scale-up the default warehouse size to run the Java UDF at scale across all cores available on all nodes on a 2XL warehouse (64 nodes). This can be done easily and quickly because of Snowflake's instant elasticity:

```
alter warehouse quickstart set warehouse_size = xxlarge;
```

Now let's run the following query to store the named entities into a table.

```
create or replace table email_named_entities_base as
select
  relative_path
  , upper(replace(get(split(relative_path, '/'), 0), '\'', '')) as mailbox
  , parseText('@email_stage_internal/' || relative_path) as named_entities
from (
  select relative_path
  from directory(@email_stage_internal)
  group by relative_path
);
```

After running this query, we can set the `warehouse` size back to a smaller size.

THIS STEP IS VERY IMPORTANT NOT TO EXHAUST YOUR TRIAL CREDIT

```
alter warehouse quickstart set warehouse_size = xsmall;
```

Verify with the following command that the warehouse is back to an `xsmall` size.

```
show warehouses;
```

The output should show the `QUICKSTART` size as being `XSMALL`. Now let's query the base table containing the named entities.

```
select
  *
from email_named_entities_base
limit 5;
```

For each email, we now have created a `MAILBOX` attribute, as well as JSON string containing the named entities present in the file.


```

167
168 | select
169 | *
170 | from email_named_entities_base
171 | limit 5;|
172

```

	RELATIVE_PATH	MAILBOX	NAMED_ENTITIES
1	dean-c/inbox/608.	DEAN-C	{"Persons":["Steiner","David","Dietrich","Dan","Dave Steiner"],"Locations":["Houston","Portland","Craigl
2	kaminski-v/inbox/637.	KAMINSKI-V	{"Persons":["Vince J","Vince JInbox","Tony Sent","Barrett","Misty Subject","Anthony Cox"],"Locations":}
3	jones-t/inbox/742.	JONES-T	{"Persons":["Sara","Jones","Tana","Koehler","Anne","Heard","Marie"],"Locations":["Houston"]}
4	tycholz-b/inbox/144.	TYCHOLIZ-B	{"Persons":["Ward","Kim S","Anne","Richard","Barry","Anne","Kim"],"Locations":["Palo Alto","Anne","Palc
5	martin-t/inbox/140.	MARTIN-T	{"Persons":["Martin","Thomas","Brian","Bryan","Gary","Thomas","Gary"],"Locations":["Texas"]}

Exploring the Mailbox Corpus

We have now extracted the named entities the analysts are interested in seeing to do some analytics on this email corpus. We can use Snowflake native capabilities to easily store and query semi-structured data, in this case JSON.

We will first create a view to parse the `NAMED_ENTITIES` JSON string and extract separately the persons and the locations entities, as well as count the number of entities identified in each email:

```

create or replace view email_info_v
as
with named_entities
as (
    select
        relative_path
        , mailbox
        , parse_json(named_entities) as named_entities
    from email_named_entities_base
)
select
    relative_path
    , mailbox
    , named_entities:Persons::variant as persons
    , array_size(persons) as num_person_entities
    , named_entities:Locations::variant as locations
    , array_size(locations) as num_location_entities
    , array_size(persons) + array_size(locations) as total_entities
    , build_scoped_file_url(@email_stage_internal, relative_path) as
scoped_email_url
from named_entities;

```

We can query the view and examine the output. Notice that we now have JSON arrays in separate columns for name and location entities, count of entities, and a scoped URL to access the file if we need to further examine its contents.

```

select * from email_info_v limit 10;

```

```

198 select
199     relative_path
200     , mailbox
201     , named_entities::Persons::variant      as persons
202     , array_size(persons)                  as num_person_entities
203     , named_entities::Locations::variant    as locations
204     , array_size(locations)                 as num_location_entities
205     , array_size(persons) + array_size(locations) as total_entities
206 from named_entities;
207
208 | select * from email_info_v limit 10;
209

```

	RELATIVE_PATH	MAILBOX	PERSONS	NUM_PERSON_ENTITIES	LOCATIONS
1	taylor-m/inbox/365.	TAYLOR-M	["Woods", "Trevor", "John", "Taylor", "Michael E", "Michael EInbox", "Woods"	12	["Woods", "Woods"]
2	white-s/inbox/143.	WHITE-S	["Evelyn", "White", "Stacey", "Stacey", "Bryce Sent", "Evelyn Cc", "Margare"	7	["Stacey"]
3	shackleton-s/inbox/493.	SHACKLETON-S	["Taylor", "Mark E", "Sara", "Sara Sent", "Taylor", "Mark E", "Bailey", "Susa"	10	["Houston"]
4	steffes-j/inbox/418.	STEFFES-J	["Jo Ann Scott", "James", "Neil Shockey", "Rich Text Formats"]	4	["California", "California", "Ca"
5	campbell-l/inbox/1232.	CAMPBELL-L	["Campbell", "Larry", "Campbell L", "Larry", "Merry Christmas", "Andrews Tex"	6	["Andrews", "Team"]
6	schoolcraft-d/inbox/17.	SCHOOLCRAFT-D	["Nancy", "Floyd", "Eric", "Carr", "James", "Blair", "Jean", "Jerry", "Clapper"	14	["Washington"]
7	nemec-g/inbox/1138.	NEMEC-G	["Jeff Hodge", "John", "Nemec", "Gerald", "Gerald Privileged", "Jeff Hodge"]	6	["Houston", "Texas"]
8	sager-e/inbox/79.	SAGER-E	["Hansen", "Leslie", "Elizabeth", "Hansen", "Leslie", "Elizabeth", "Thanks Le"	7	["Elizabeth"]
9	mccarty-d/inbox/65.	MCCARTY-D	["Susan", "McCarty", "Danny", "Armstrong", "Julie", "Cindy", "Stan", "Cindy"	8	["Omaha"]
10	hyatt-k/inbox/12.	HYATT-K	["Eric", "Kevin", "Kevin", "Kim", "W"]	5	["Sun", "Salt River", "Sun Dev"

[[PERSONS
[
 "Woods",
 "Trevor",
 "John",
 "Taylor",
 "Michael E",
 "Michael EInbox",
 "Woods",
 "Trevor Sent",
 "Taylor",
 "Michael E Subject",
 "Mike",
 "T"
]
]
]

We can now query the view to retrieve various entity metrics. For example, the following query identifies the top 5 emails in terms of total number of entities.

```

select
    relative_path
    , mailbox
    , total_entities
    , num_person_entities
    , num_location_entities
    , persons
    , locations
    , scoped_email_url
from email_info_v
order by total_entities desc
limit 10;

```

```

191 | select
192     relative_path
193     , mailbox
194     , total_entities
195     , num_person_entities
196     , num_location_entities
197     , persons
198     , locations
199 from email_info_v
200 order by total_entities desc
201 limit 5;

```

	RELATIVE_PATH	MAILBOX	TOTAL_ENTITIES	NUM_PERSON_ENTITIES	NUM_LOCATION_ENTITIES	PERSON
1	williams-j/inbox/38.	WILLIAMS-J	16	15	1	["Murre"
2	nemec-g/inbox/106.	NEMEC-G	16	14	2	["Jeffre"
3	schoolcraft-d/inbox/17.	SCHOOLCRAFT-D	15	14	1	["Nanc"
4	white-s/inbox/36.	WHITE-S	15	14	1	["Andy"
5	jones-t/inbox/388.	JONES-T	14	13	1	["Gray"

The following query aggregates statistics on the number of entities identified per mailbox and returns the top 5 mailbox by total number of entities identified in the mailbox.

```

select
    mailbox

```

```

, count(relative_path)          as num_emails
, sum(total_entities)           as sum_entities
, round(avg(total_entities))    as avg_entities
, sum(num_person_entities)      as sum_persons
, round(avg(num_person_entities)) as avg_persons
, sum(num_location_entities)    as sum_locations
, round(avg(num_location_entities)) as avg_locations
from email_info_v
group by mailbox
order by sum_entities desc
limit 5;

```

We know that the most prolific mailbox is `WHITE-S` for identified entities in this email corpus.

Performing Analytics on the Mailbox

Let's assume an analyst wants to identify all email correspondence where the name of 'Willman' (chosen completely randomly in this example) is mentioned. With Snowflake, you can easily flatten arrays to run this type of query.

```

select
    relative_path
    , mailbox
    , pers.value::string as person_entity
from
    email_info_v
    , lateral flatten(input => Persons) pers;

```

```

216
217 | select
218 |     relative_path
219 |     , mailbox
220 |     , pers.value::string as person_entity
221 | from
222 |     email_info_v
223 |     , lateral flatten(input => Persons) pers;

```

Objects Query Results Chart

	RELATIVE_PATH	MAILBOX	PERSON_ENTITY
1	farmer-d/inbox/55.	FARMER-D	Sally Beck Charity Golf Event
2	farmer-d/inbox/55.	FARMER-D	Robert
3	farmer-d/inbox/55.	FARMER-D	Charles
4	farmer-d/inbox/55.	FARMER-D	Daren
5	farmer-d/inbox/55.	FARMER-D	Smith
6	farmer-d/inbox/55.	FARMER-D	George

We can use the previous query as a CTE to retrieve all emails mentioning a specific person, including a scoped URL to access the actual email file.

```

with persons_flattened as (
    select
        relative_path
        , mailbox
        , pers.value::string as person_entity
    from
        email_info_v

```

```

        , lateral flatten(input => Persons) pers
    )
select
    relative_path
    , mailbox
    , person_entity
    , build_scoped_file_url(@email_stage_internal,relative_path) as scoped_email_url
from persons_flattened
where person_entity like '%Willmann%';

```

```

224
225 with persons_flattened as (
226     select
227         relative_path
228         , mailbox
229         , pers.value::string as person_entity
230     from
231         email_info_v
232         , lateral flatten(input => Persons) pers
233     )
234     select
235         relative_path
236         , mailbox
237         , person_entity
238         , build_scoped_file_url(@email_stage_internal,relative_path) as email_url
239     from persons_flattened
240     where person_entity like '%Willmann%';

```

Objects Query Results Chart

	RELATIVE_PATH	MAILBOX	PERSON_ENTITY	EMAIL_URL
1	may-l/inbox/503.	MAY-L	Donnie Willmann	https://uua85655.snowflakecomputing.com/api/files/01a22a34-0401-279d-003d-7
2	smith-m/inbox/148.	SMITH-M	Donnie Willmann	https://uua85655.snowflakecomputing.com/api/files/01a22a34-0401-279d-003d-7
3	keavey-p/inbox/82.	KEAVEY-P	Donnie Willmann	https://uua85655.snowflakecomputing.com/api/files/01a22a34-0401-279d-003d-7
4	donohoe-t/inbox/93.	DONOHOE-T	Donnie Willmann	https://uua85655.snowflakecomputing.com/api/files/01a22a34-0401-279d-003d-7
5	beck-s/inbox/307.	BECK-S	Donnie Willmann	https://uua85655.snowflakecomputing.com/api/files/01a22a34-0401-279d-003d-7

Click on any MAY-L mailbox email to download and review the email. This will allow you to learn a little bit more about that person's role and responsibilities in the Enron organization.

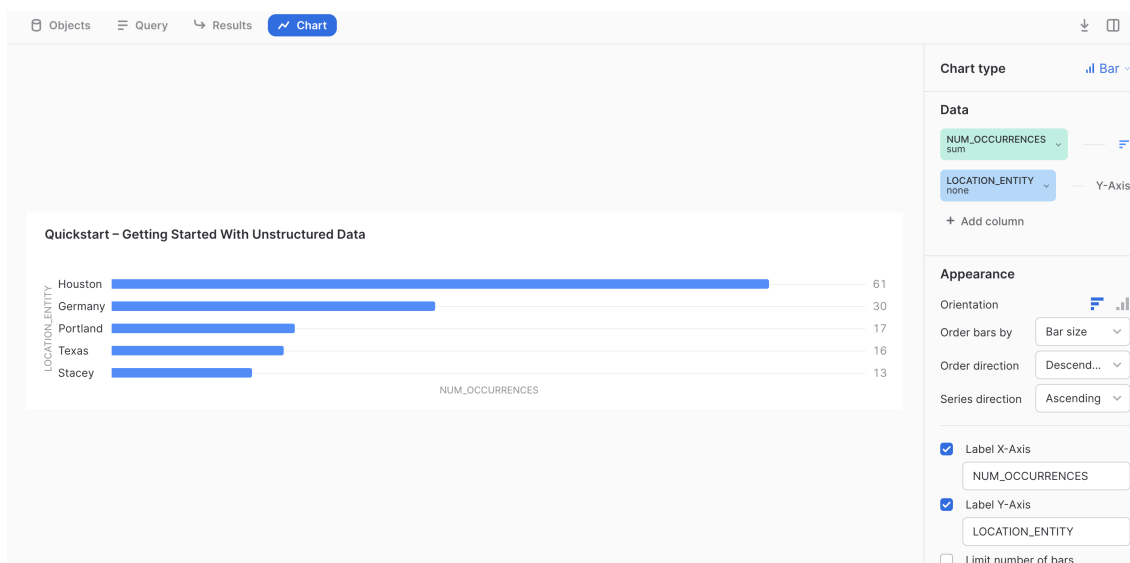
Let's assume we want to identify the top 5 locations mentioned in the email corpus.

```

with locations_flattened as (
    select
        relative_path
        , mailbox
        , loc.value::string as location_entity
    from
        email_info_v
        , lateral flatten(input => Locations) loc
)
select
    location_entity
    , count(location_entity) as num_occurrences
from locations_flattened
group by location_entity
order by num_occurrences desc
limit 5;

```

We can use Snowsight to produce a visualization. Click on chart and set the settings as shown below.



As you can see, we can perform aggregations, and analytics on unstructured text data after extracting entities and information of interest. At this point, one could run more advanced data science use cases or visualizations using the numerous options available in Snowflake.

Share Unstructured Data

In this example, we want to share all the email corpus mentioning 'Willman' with another party using a Snowflake reader account.

Creating a Reader Account

The first step is to create a reader account as follows. Note that you will need to provide a secure password of your choice.

```
use role accountadmin;
create managed account if not exists emaildb_reader
admin_name='admin', admin_password='<password>',
type=reader, COMMENT='Emaildb Reader Account';
```

This command should return the account name, and the URL to access the account. Please copy and paste the output of the command as it provides you the login URL information for the account, as well as the user and password you chose

```
{
  "accountName": "<account_name>",
  "loginUrl": "https://<account>.snowflakecomputing.com"
}
```

You can run the following command to retrieve the managed account information at anytime.

```
show managed accounts;
```

You can connect to the previous reader account using the URL, and the userid/password credentials you passed as parameters.

Creating a Secure View to Share

Let's now create the secure view based on the query ran in the previous section that will be shared with the reader account.

```
use role sysadmin;
use schema emaildb.raw;

create or replace secure view email_corpus_willman_v as
with persons_flattened as (
  select
    relative_path
    , mailbox
    , pers.value::string as person_entity
  from
    email_info_v
    , lateral flatten(input => Persons) pers
)
select
  relative_path
  , mailbox
  , person_entity
  , build_scoped_file_url(@email_stage_internal,relative_path) as email_url
from persons_flattened
where person_entity like '%Willmann%';
```

Create the Share

We can now create the share as follows. You will need to provide the reader account name created earlier:

```
-- Create the share object
use role accountadmin;
create or replace share email_corpus
comment='Share in scope email corpus information';

--what are we sharing?
grant usage on database emaildb to share email_corpus;
grant usage on schema emaildb.raw to share email_corpus;
grant select on view emaildb.raw.email_corpus_willman_v to share email_corpus;

-- whom are we sharing with?
alter share email_corpus add accounts = <reader-account-locator>;
```

We can review the share we have just created. The following command provides all the shares in the account.

```
-- check the share
show shares like 'email_corpus';
```

We can get more details about the share, and the scope of the objects shared using the following command.

```
-- review share
describe share email_corpus;
```

```

314
315 -- review share
316 describe share email_corpus;

```

Objects Query Results Chart			
	kind	name	shared_on
1	DATABASE	EMAILDB	2022-02-07 14:01:06.307 -0800
2	SCHEMA	EMAILDB.RAW	2022-02-07 14:01:06.604 -0800
3	VIEW	EMAILDB.RAW.EMAIL_CORPUS_WILLMAN_V	2022-02-07 14:01:07.077 -0800

This command shows us that the view `EMAIL_CORPUS_WILLMAN_V` is shared from the database `EMAILDB` and schema `RAW` in the current account.

Accessing Shared Data

Switch to the web browser tab where you opened the session with the reader account created in step 6.1 or open a new session on the reader account. Now, click on the blue Snowsight UI button at the top and authenticate again.

After logging in, as this is a new account, click on the worksheet button at the top and create a new virtual warehouse.

```

use role sysadmin;

create or replace warehouse compute_wh
warehouse_size=xsmall
auto_suspend=1
auto_resume=true
initially_suspended=true;

grant usage on warehouse compute_wh to public;

```

First, let's switch back to the `ACCOUNTADMIN` role. Click on the **Home** button in the top-left. Then in the top-left, click on **ADMIN**, then hover over **Switch Role**, and click on **ACCOUNTADMIN**.

A

ADMIN

SYSADMIN

▼

A

ADMIN

Running latest version in canary

Switch to main

Switch Role

👤

SYSADMIN

>

Profile


Partner Connect

Documentation [↗](#)

Sign Out

Worksheets

RecentShared with meMy WorksheetsF

TITLE
 Getting Started with Unstructured Dat

🔍 Roles

👤

SYSADMIN

✓

👤

ACCOUNTADMIN

👤

PUBLIC

👤

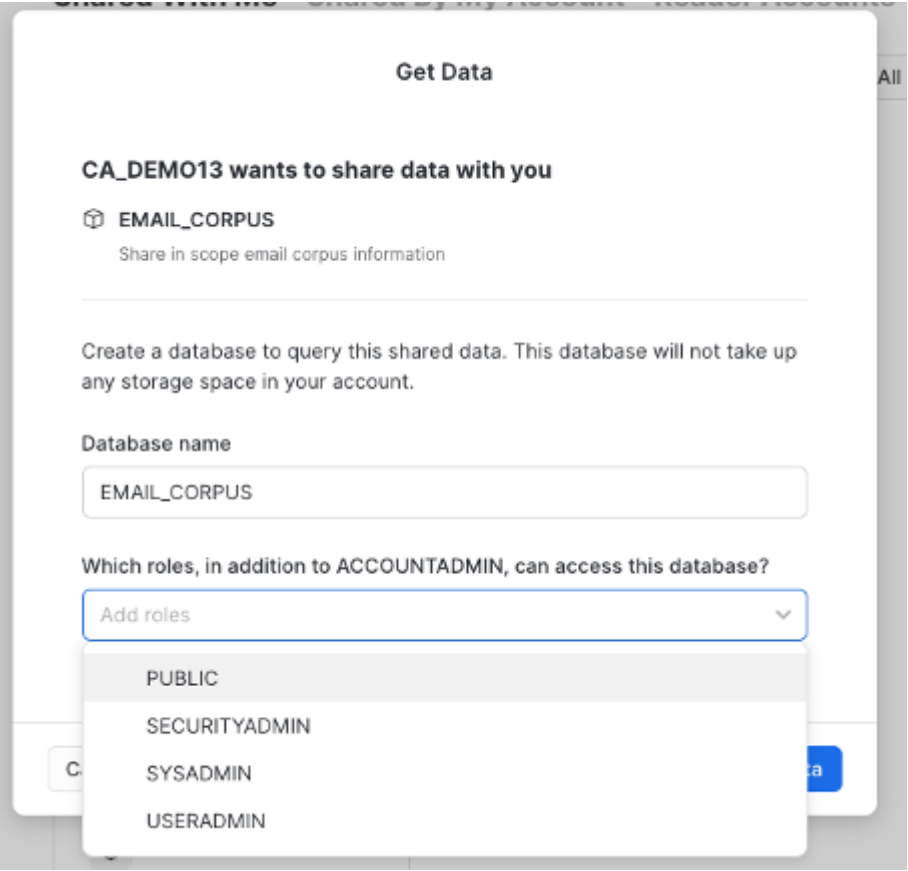
SECURITYADMIN

👤

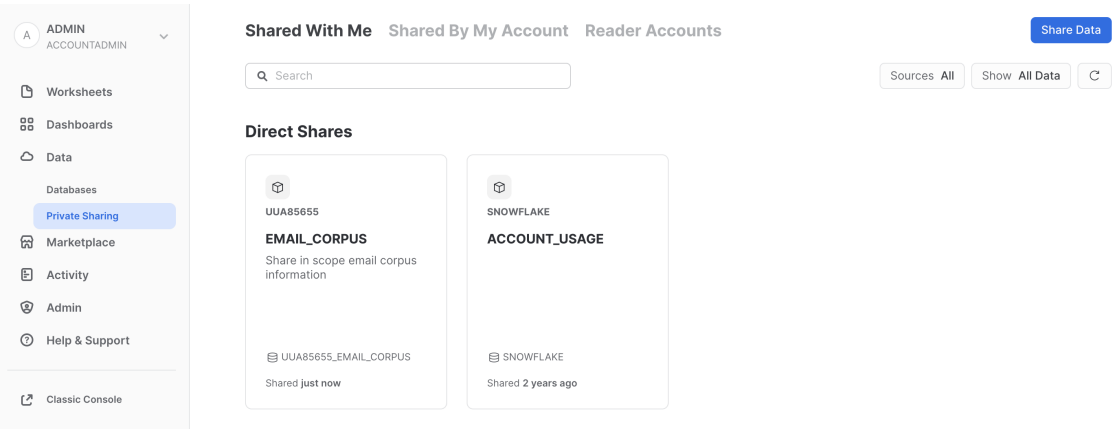
USERADMIN

Now let's view the shared data. In the pane on the left, click on on **Data**, then **Private Sharing**. You will see the `EMAIL_CORPUS` database listed under **Ready to Get**. Select it and give the database name `EMAIL_CORPUS` and

make it available to `PUBLIC` , then click the **Get Data** button.



Click on **Databases**, then click on the **Refresh** button (round arrow button on the right side above the database list). You will now see the database `EMAIL_CORPUS` .



Select the worksheet created in this reader account. Add the following commands to set the worksheet session parameters and review the view objects available.

```
use role sysadmin;
use schema <account-locator>_email_corpus.raw;
use warehouse compute_wh;
```

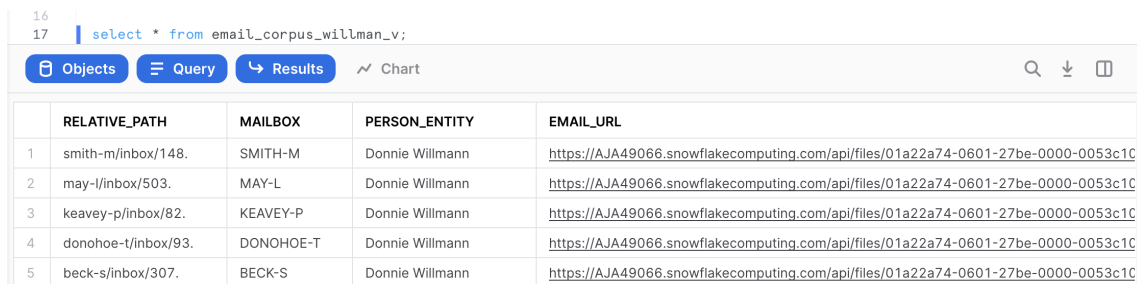
```
show views;
```

If you expand the database hierarchy on the left side of the window, you will see that the share appears from the consumer side as a database `EMAIL_CORPUS`, with a schema `RAW` and a single view object `EMAIL_CORPUS_WILLMAN_V`.

We can now query the shared data. The query below will display only the emails related to 'Willmann'.

```
select * from email_corpus_willman_v;
```

From the results, notice that all the URLs are encrypted, not revealing any information of the location where the shared data came from. Click on any `EMAIL_URL` and get access to the actual email text downloaded to your workstation. Download and review the email. Make sure it is valid.



The screenshot shows a Snowflake query results window. The query bar contains the text `select * from email_corpus_willman_v;`. Below the query bar, there are tabs for 'Objects', 'Query', 'Results' (which is selected), and 'Chart'. The 'Results' tab displays a table with 5 rows and 4 columns: `RELATIVE_PATH`, `MAILBOX`, `PERSON_ENTITY`, and `EMAIL_URL`. Each row represents an email entry, with the `EMAIL_URL` column containing encrypted links.

	RELATIVE_PATH	MAILBOX	PERSON_ENTITY	EMAIL_URL
1	smith-m/inbox/148.	SMITH-M	Donnie Willmann	https://AJA49066.snowflakecomputing.com/api/files/01a22a74-0601-27be-0000-0053c1C
2	may-l/inbox/503.	MAY-L	Donnie Willmann	https://AJA49066.snowflakecomputing.com/api/files/01a22a74-0601-27be-0000-0053c1C
3	keavey-p/inbox/82.	KEAVEY-P	Donnie Willmann	https://AJA49066.snowflakecomputing.com/api/files/01a22a74-0601-27be-0000-0053c1C
4	donohoe-t/inbox/93.	DONOHOE-T	Donnie Willmann	https://AJA49066.snowflakecomputing.com/api/files/01a22a74-0601-27be-0000-0053c1C
5	beck-s/inbox/307.	BECK-S	Donnie Willmann	https://AJA49066.snowflakecomputing.com/api/files/01a22a74-0601-27be-0000-0053c1C

You have now completed this demonstration of how unstructured data can be securely shared in the Snowflake Data Cloud using Snowflake Data Sharing capabilities.

Conclusion

Congratulations! You used Snowflake to perform natural language processing on email files.

What we've covered

- Accessing external data with an **External Stage**
- Storing unstructured data with an **Internal Stage** and **SnowSQL**
- Governing unstructured data with **Role-Based Access Control**
- Catalog unstructured data with **Directory Tables**
- Securely access unstructured data with **Scoped, File, and Pre-signed URLs**
- Processing unstructured data with a **Java UDF**
- Sharing unstructured data in the **Data Cloud**