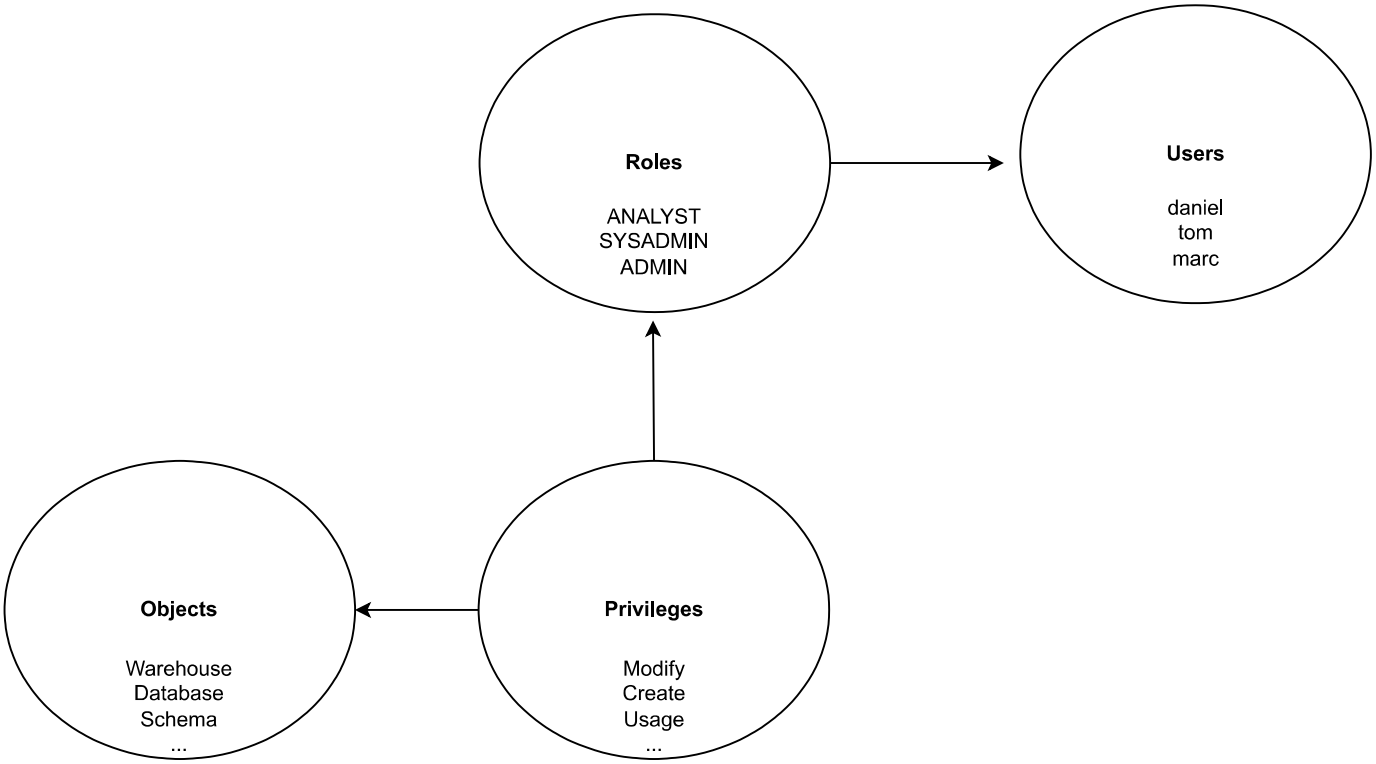
Snowflake

Snowflake Roles

- Users get Assigned Roles

- Roles hold Privileges to operate on Objects

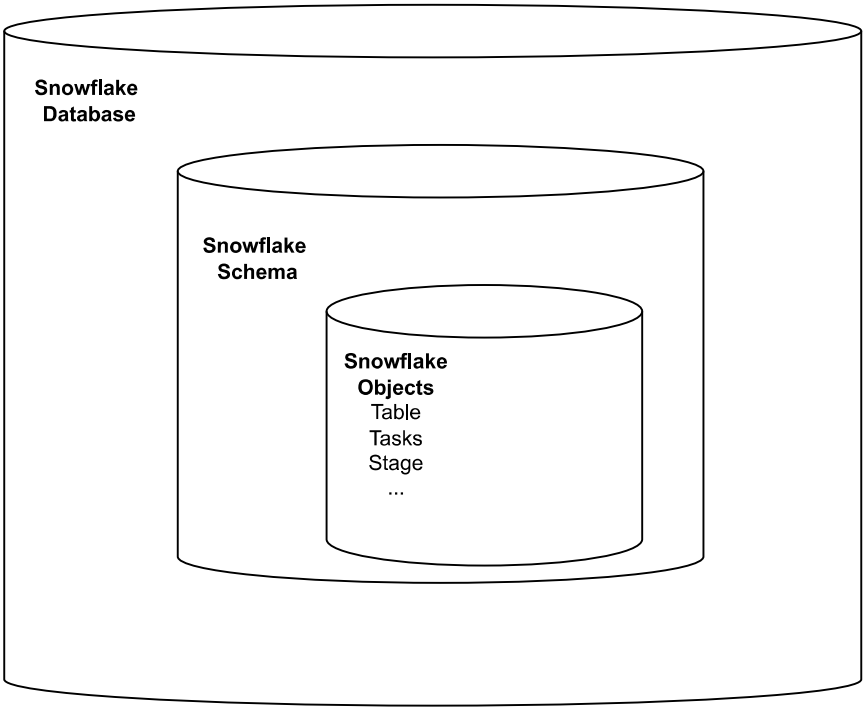



```
GRANT ROLE ANALYST TO USER john;

GRANT USAGE ON DATABASE DEMO TO ROLE ANALYST;
```

Snowflake Object Hierarchy

Most Snowflake Objects are contained withing a Schema - which itself is cointained within a Database.

To give select privilege to a table, you 'll need to give access to the Database and Schema where the table is located

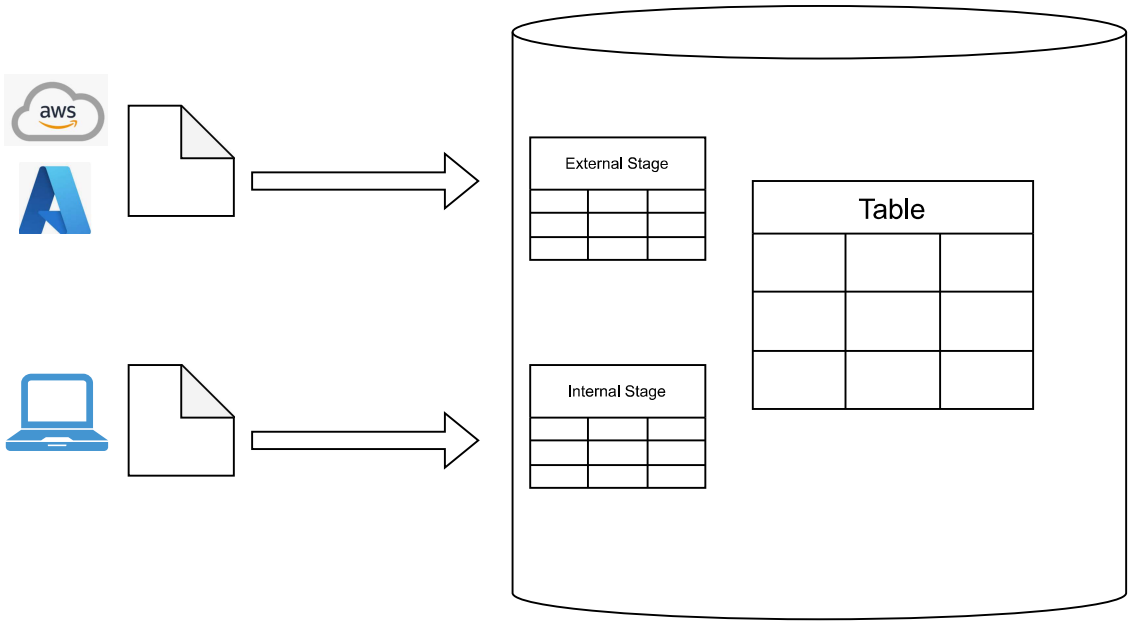



```
GRANT SELECT ON TABLE DEMO.DEMO_SCHEMA.TABLE_1 TO ROLE MGR;

GRANT USAGE ON DATABASE DEMO TO ROLE MGR;

GRANT USAGE ON SCHEMA DEMO.DEMO_SCHEMA TO ROLE MGR;
```

Snowflake Stages

Stages specify where the data is stored before getting loaded into a Table.

There are two types of stages: Internal stage - for loading data from your laptop - and External stage - for loading from the Cloud



Examples:

To create an Internal stage - simply by not specifying an URL:

```
CREATE STAGE DEMO.DEMO_SCHEMA.INTERNAL_STAGE;
```

To create an External stage - linked to a public bucket:

```
CREATE STAGE DEMO.DEMO_SCHEMA.EXTERNAL_STAGE
URL='s3://mypublicbucket/mypublicfolder';
```

To see the content of a stage:

```
LIST @DEMO.DEMO_SCHEMA.INTERNAL_STAGE;
```

To create a stage containing CSV files:

```
CREATE STAGE DEMO.DEMO_SCHEMA.INTERNAL_STAGE
FILE_FORMAT=(TYPE=CSV);
```

To query the first 2 columns of a stage formatted as CSV:

```
SELECT

t.$1,
t.$2

FROM @DEMO.DEMO_SCHEMA.INTERNAL_STAGE t;
```

To create an External stage - linked to a private bucket:

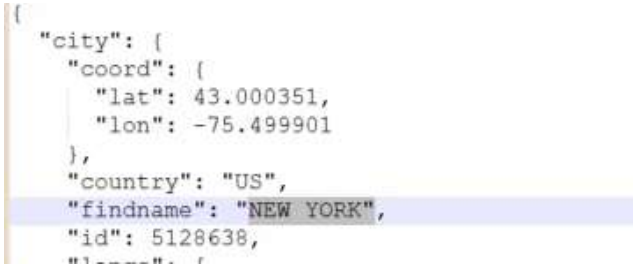
(Configuration of the private bucket is detailed further down)

```
CREATE STAGE DEMO.DEMO_SCHEMA.EXTERNAL_STAGE
URL='s3://myprivatebucket/myprivatefolder'
STORAGE_INTEGRATION= S3_INT;
```

To select specific fields from a stage containing JSON payload:

When the stage is formatted as JSON, everything is contained in the first column selected

```
SELECT

t.$1:city:findname,
t.$1:city:coord:lat

FROM @DEMO.DEMO_SCHEMA.WEATHER_STAGE t;
```



To access an array element within your JSON payload

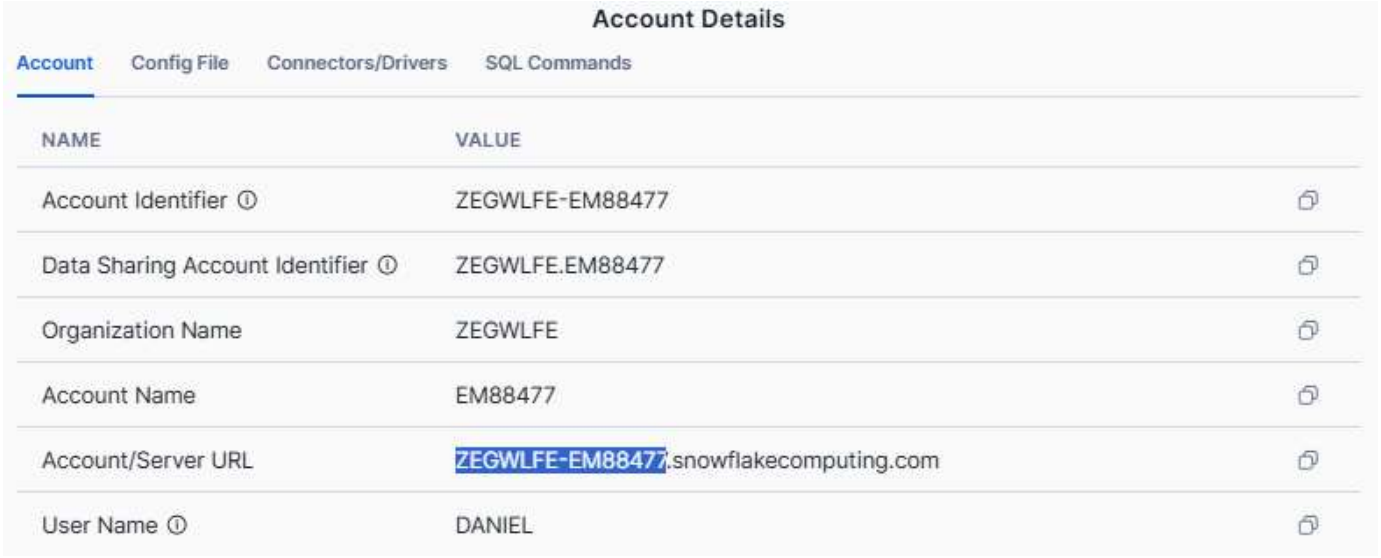
Specify the index with Squared Brackets [ ]


```
SELECT

t.$1:weather[0]:description

FROM @DEMO.DEMO_SCHEMA.WEATHER_STAGE t;
```

Snowsql

Snowsql is a command line tool that allows you to connect to Snowflake and put files from your local drive into a Snowflake Internal stage.

To connect to it, open your command line and type: **snowsql -a** and paste the first part of your Snowflake account URL in **Account Details**




```
C:\Users\d.weigel>snowsql -a ZEGWLFE-EM88477
```

Once connected with User / Password, precise the context of the stage you want to put files + the Warehouse you intend to use

```
Type SQL statements or !help
daniel#COMPUTE_WH@(no database).(no schema)>USE WAREHOUSE COMPUTE_WH;
```

```
1 Row(s) produced. Time Elapsed: 0.063s
daniel#COMPUTE_WH@(no database).(no schema)>USE DATABASE DEMO;
```

```
1 Row(s) produced. Time Elapsed: 0.094s
daniel#COMPUTE_WH@DEMO.PUBLIC>USE SCHEMA DEMO_SCHEMA;
```

Use the **PUT command** to put your local file into the stage.

Tips:
- put quotes around file path to account for possible spaces in your path
- If it's a Windows machine, don't forget to replace '\' by '/' in your path
- If you want to reupload the same file, add the option **OVERWRITE=TRUE**
- Within a folder, **use the wildcard *** to match a specific pattern and upload many files at once (example *.csv for all csv files)

```
EMO.DEMO_SCHEMA>PUT 'file://C:/Users/daniel/UdemyFile.xlsx' @INTERNAL_STAGE;
```

The COPY INTO command

Once you got your files into your stage, use the COPY INTO command to copy them over to your final table

Once you have refined your stage select statement paste it surrounded by () after the FROM clause of the COPY INTO command

```
COPY INTO DEMO.DEMO_SCHEMA.CITY
FROM (

SELECT
t.$1:city:findname
FROM @DEMO.DEMO_SCHEMA.WEATHER_STAGE t
)

ON_ERROR= SKIP_FILE_3
;
```

The **ON_ERROR** option allows you to control, what happens if theres an error found with a file being copied:

- **CONTINUE**: keeps on going no matter the number of files in error

- **SKIP_FILE**: controls the number of failed copy files before the COPY INTO statement throws an error