Snowflake SnowPro Certification Exam Cheat Sheet by Jeno Yamma

// Snowflake - General //

Introduction

- Analytic data warehouse
- SaaS offering
  - No hardware (or installation or updates/patch)
  - No ongoing maintenance or tuning
  - Can't run privately (on-prem or hosted)
- Runs completely on the cloud
  - Either AWS, Azure or GCP (NO ON-PREM or HYBRID!)
  - Has its own VPC!
- Decoupled compute and storage (scaled compute does not need scaled storage)

Pricing

- Unit costs for Credits and data storage determined by region (not cloud platform)
  - Cost of Warehouse used
    - Minimum 60s on resuming, per 1s after
  - Cost of storage (Temp, Transient, Perm Tables & time-travel data & fail-safe)
    - Calcs based on daily average
- Pricing model = on demand or discounted upfront
- Just remember:
  - >= Enterprise = 90 days time travel (default: 1 day for all) + materialized view + multi-cluster warehouse
  - >= Business Critical = lots more security (HIPAA, SOC 1&2, PCI DSS)
  - VPS edition = own cloud service layer (not shared with accounts)

Supported Regions

- Multi-region account isn't supported, each SF in single region
- 17 Regions in total
  - AWS - 9 regions (Asia Pacific-3, EU-2, North America-4)
  - GCP - 1 (Asia Pacific-0, EU-0, North America-1) >> in preview
  - Azure - 7 (Asia Pacific-2, EU-1, North America-4)

// Connecting to SF //

	Create	Clone	Drop	Modify	Load Data	Transfer Ownership
Tables	✔	✔	✔		✔	✔
Views	✔		✔			✔
Schemas	✔	✔	✔			✔
Stages	✔	✔	✔	✔		✔
File Formats	✔	✔	✔	✔		✔
Sequences	✔	✔	✔	✔		✔

- Web-based UI (see above for usage, capabilities, restriction)
- Command line client (SnowSQL)
- ODBC and JDBC (you have to download the driver!)
- Native Connectors (Python, Spark & Kafka)
- Third party connectors (e.g. Matilion)
- Others (Node.js, .Net)

Snowflake Address

`https://account_name_region_{provider}.snowflakecomputing.com`

Account_name either:

- AWS = account_name.region
- GCP/Azure = account_name.region.gcp/azure

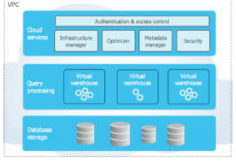
e.g. https://pp12345.ap-southeast-2.snowflakecomputing.com

// Architecture //

Overview

- Hybrid of shared-disk db and shared-nothing db
  - Uses central data repo for persisted storage - All compute nodes have data access
  - Using MPP clusters to process queries - Each node stores portion of data locally

Architectural layers



Storage

- All data stored as an internal, optimised, compressed columnar format (micro-partitions - represents logical structure of table)
- Can't access the data directly, only through Snowflake (SQL etc)

Query Processing

- Where the processing of query happens
- Uses virtual warehouses - an MPP compute cluster
  - Each cluster = Independent - doesn't share resources with other vwh and doesn't impact others

Cloud Services

- Command centre of SF - coordinates and ties all activities together within SF
- Gets provisioned by Snowflake and within AWS, Azure or GCP
  - You don't have access to the build of this or to do any modifications
  - Instance is shard to other SF accounts unless you have VPS SF Edition
- Services that this layer takes care of:
  - Authentication
  - Infrastructure management
  - Metadata management
  - Query parsing and optimisation
  - Access control

Caches

- Snowflake Caches different data to improve query performance and assist in reducing cost

>Metadata Cache - Cloud Services layer

- Improves compile time for queries against commonly used tables

>Result Cache - Cloud Services layer

- Holds the query results
- If Customers run the exact same query within 24 hours, result cache is used and no warehouse is required to be active

>Local Disk Cache or Warehouse Cache - Storage Layer

- Caches the data used by the SQL query in its local SSD and memory
- This improves query performance if the same data was used (less time to fetch remotely)
- Cache is deleted when the Warehouse is suspended

// Data Loading //

File Location
- On Local
- On Cloud
  - AWS (Can load directly from S3 into SF)
  - Azure (Can load directly from Blob Storage into SF)
  - GCP (Can load directly from GCS into SF)

File Type
- Structured
  - Delimited files (CSV, TSV etc)
- Sem-structured
  - JSON (SF can auto detect if Snappy compressed)
  - ORC (SF can auto detect if Snappy compressed or zlib)
  - Parquet (SF can auto detect if Snappy compressed)
  - XML (in preview)
- * Note: If files = uncompressed, on load to SF it is gzip (can disable)

>Compression
- SF auto compresses data from local fs to gzip - can change (AUTO_COMPRESS)
- Specify compression type on loading compressed data

>Encryption for load
- Loading unencrypted files
  - SF auto encrypts files using 126-bit keys! (or 256-keys - requires configuring)
- Loading encrypted files
  - Provide your key to SF on load

Best Practice

>File Sizing
- ~10-100 MB file, compressed - Optimises parallel operations for data load
- Aggregate smaller files - Minimises processing overhead
- Split up larger files to small files - Distributes workload among server in active vwh
- # data files processed in parallel depends on capacity of servers in vwh
- Parquet: >3GB compressed could time out - split into 1GB chunks

>Semi Structured Sizing
- VARIANT data type has 16 MB compressed size limit per row
- For JSON or Avro, outer array structure can be removed using *STRIP_OUTER_ARRAY*.

>Continues Data Loads File Sizing
- Load new data within a minute after file notification sent, longer if file is large
  - Large compute is required (decompress, decrepit, transform etc).
- If > 1min to accumulate MBs of data in source app, create data file once per min
  - Lowers cost, improve performance (load latency)
- Creating smaller files and staging them in cloud storage >1 time per minute = disadvantage
  - Reduction in latency between staging and loading data != guaranteed
  - Overhead to manage file in internal load queue - increase utilisation cost
  - Overhead charges apply if per than 1000 files in queue

Planning Data Load
Dedicating Separate Warehouse
- Load of large data = affect query performance
  - Use seperate Warehouse
  - # data files processed in parallel determined by servers in the warehouse.
  - Split large data to scale linearly, use of small vwh should be sufficient

Staging Data

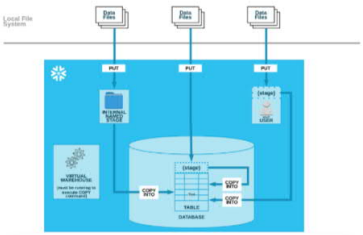
> Organising Data by Path - Snowflake Staging
- Both internal and external ref can include path (prefix) in the cloud storage
- SF recommends (for file organisation in Cloud Storage):
  1. Partition into logical path - includes identifying detail with date
  2. Organising file by path = allow copy fraction of partitioned data in SF in one command
     - Result: Easier to execute concurrent COPY statements - takes advantage of parallel operations
- Named stage operation:
  - Staging = A place where the location/path of the data is stored to assist in processing the upload of files
  - Remember: Files uploaded to snowflake Staging area using PUT are automatically encrypted with 128-bit or 256-bit *(*CLIENT_ENCRYPTION_KEY_SIZE to specify) keys!

Loading Data
> COPY command - parallel execution
- Supports loading by Internal Stage or S3 Bucket path
- Specifying specific list of files to upload (1000 files max at a time)
- Identify files through pattern matching (regex
  - Validating Data:
    - Use VALIDATION_MODE - validate errors on load - does not load into table
    - ON_ERROR to run actions to follow
- When COPY command runs, SF sets load status in the table's metadata
  - Prevents parallel COPY from loading the same file
  - On complete, SF sets the load status and to those that failed as well
  - Metadata (expires after 64 days):
    - Name of file
    - File size
    - ETag of the file
    - # rows parsed in file
    - Timestamp of last load
    - Information on errors during load
- SF recommends removing the Data from the Stage once the load is completed to avoid reloading again - use REMOVE command (and specify PURGE in COPY argument)
  - Improves performance - doesn't have to scan processed files

> Note for Semi-structured
- SF loads ss data to VARIANT type column
- Can load ss data into multiple columns but ss data must be stored as field in structured data
- Use FLATTEN to explode compounded values into multiple rows

>Data Transformation during load
- Supported
  - Sequence, substring, to_binary, to_decimal
- Commands not supported
  - WHERE, FLATTEN, JOIN, GROUP BY, DISTINCT (not fully supported)
  - VALIDATION_MODE (if any aggregation applied)
  - CURRENT_TIME (will result in the same time)

Snowflake Stage



Types of Stages

- Default: each table and user are allocated an internal named stage
  - Specify Internal Stage in PUT command when uploading file to SF
  - Specify the Same Stage in COPY INTO when loading data into a table

> User Stage (Ref: @~)

- Accessed by single user but need copying to multiple tables
- Can't be altered or dropped
- Can't set file format, need to specify in COPY command to table

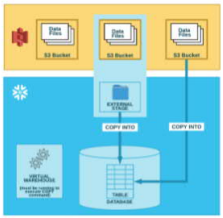
> Table Stage (Ref: @%)

- Accessed by multiple users and copied to single tale
- Can't be altered or dropped
- Can't set file format, need to specify in COPY command to table
- No transformation while loading

Internal Named Stages (Ref: @)

- A Database object
- Can load data into any tables (Needs user with privilege)
- Ownership of stage can be transferred

AWS - Bulk Load Loading from S3



- SF uses S3 Gateway Endpoint. If region bucket = SF region - no route through public internet.

>Securely access S3

1. Configure storage integration object - delegate authentication responsibility for external cloud storage to SF identity and IAM
2. Configure AWS IAM role -> allow S3 bucket access
3. Configure IAM user, provide Key and Secret Key

Snowpipe - Incremental Load

>Introduction

	COPY Command	Snowpipe
Authentication	Security options supported by the client for authenticating and initiating a user session.	***When calling the REST endpoints:*** Key pair authentication with JSON Web Token (JWT). JWTs are signed using a public/private key pair with RSA encryption.
Transactions	Adds data to a table in transactions alongside any other SQL statements submitted manually by users.	Adds data to a table in transactions controlled by Snowflake with no opportunity to involve other statements in the transaction.
Load History	Stored in the metadata for the target table for 64 days. Available upon completion of the COPY statement as statement results.	Stored in the metadata for the pipe for 14 days. Must be requested from Snowflake (via a REST endpoint, SQL table function, or ACCOUNT_USAGE view).
Compute Resources	Requires a user-specified warehouse to execute COPY statements.	Uses Snowflake-supplied compute resources.
Cost	Billed for the amount of time each virtual warehouse is active.	Billed according to the compute resources used in the Snowpipe warehouse while loading the files.

- Enable you to loads data as soon as they are in stage (uses COPY command)
- Uses Pipe (first-class SF object) which contains the COPY command
  - Can DROP, CREATE, ALTER, DESCRIBE and SHOW PIPES
- Think of this as SF way of streaming data into the table as long as you have created a Named Stage.
- Generally loads older files first but doesn't guarantee - Snowpipe appends to queue.
- Has loading metadata to avoid duplicated loads

>Billing and Usage

- User doesn't need to worry about managing vwh
- Charges based on resource usage - consume credit only when active
- View charges and usage
  - Web UI - Account > Billing & Usage
  - SQL - Information Schema > PIPE_USAGE_HISTORY

>Automating Snowpipe

- Check out their AWS, GCP, Azure, and REST API article

Querying from Staged File - Query data outside SF

- SF allows you to query data directly in external locations as long as you have created a stage for it
  - SF recommends using simple queries only for
- Performance = impacted since it's not compressed and columnised to SF standard

// Monitoring //

Resource Monitor

- Helps control the cost and unexpected spike in credit usage
- Set Actions (or triggers) to Notify and/or Suspend if credit usage is above certain threshold
- Set intervals of monitoring
- Created by users with admin roles (ACCOUNTADMIN)
- If monitor suspends, any warehouse assigned to the monitor cannot be resumed until
  1. Next interval starts
  2. Minitor is dropped
  3. Credit quota is increased
  4. Warehouse is no longer assigned to the monitor
  5. Credit threshold is suspended

// Unloading Data //

Introduction

- Allows you to export your data out of SF
- Uses COPY INTO <location> (either external or SF stage)
    - If SF then use GET command to download file
    - Use SELECT and other full SQL syntax to build data for export
    - Can specify MAX_FILE_SIZE to split into multiple files by chunks

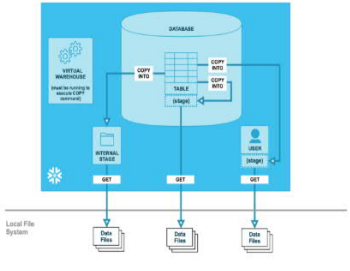
Format and Restrictions

- Allowed formats for export (only UTF-8 as encoding allowed)
    - Delimited Files (CSV etc)
    - JSON
    - Parquet
- Allowed compression for export
    - gzip (default)
    - bzip2
    - Brotli
    - Zstandard
- Encryption for export
    - Internal Stage - 128-bit or 256-bit - gets decrypted when downloaded
    - External - Customer supply encryption key

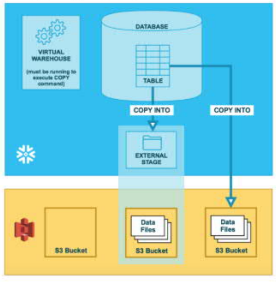
Considerations

- Empty String and Null
    - FIELD_OPTIONALLY_ENCLOSED_BY
    - EMPTY_FIELD_AS_NULL
    - NULL_IF
- Unloading a single file
    - MAX_FILE_SIZE default = 16mb
    - Max max is 5gb for AWS, GCP, and 256mb for Azure
- Unloading Relational table to JSON
    - User OBJECT_OBSTRUCT obstruct in COPY command to convert rows of relational table to VARIANT column then unload the data as per usual

Unloading in SF Stage



Unloading into S3



// Virtual Warehouse //

Cluster of resource

- Provides CPU, Mem and Temp Storage
- Is active on usage of SELECT and DML (DELETE, INSERT, COPY INTO etc)
- Can be stopped at any time and resized at any time (even while running)
    - Running queries are do not get affected, only new queries
- Warehouse sizes = T-shirt sizes (generally query performance scales linearly with vwh size)
    - X-Small - 4X-Large
- When creating a vwh you can specify:
    - Auto-suspend (default: 10mins) - suspends warehouse if active after certain time
    - Auto-resume - auto-resumes warehouse whenever a statement that requires active vwh is required
- Query Caching
    - vwh maintains cache of table data - Improves query performance
    - The Larger the vwh, larger the cache - Cache dropped when vwh suspended

Choosing VWH

- SF recommends experimenting by running the same queries on different vwh
- Monitoring Warehouse Load - WebUI: In Warehouse (view queued and running queries)

Query Processing and Concurrency

- When queries submitted, vwh calls and reserves resource
    - Query is queued if vwh doesn't have enough resource
        - STATEMENT_QUEUED_TIMEOUT_IN_SECONDS
        - STATEMENT_TIMEOUT_IN_SECONDS
        - Both can be used to control query processing and concurrency
- Size and complexity of query determines the concurrency (also # queries being executed)

Multi-cluster warehouse

- Up to 10 server clusters
- Auto-suspend and auto-resume is of whole cluster not 1 server
- Can resize anytime
- Multi-cluster mode
    - Maximised - associate same min and max for # cluster
        - SF starts all clusters at start would be good for expected workload
    - Auto-scale - different min and max for # cluster
        - To help control this, SF provides scaling policy

Policy	Description	Cluster Starts...	Cluster Shuts Down...
Standard (default)	Prevents/minimizes queuing by favoring starting additional clusters over conserving credits.	Immediately when either a query is queued or the system detects that there's one more query than the currently-running clusters can execute.	After 2 to 3 consecutive successful checks (performed at 1 minute intervals), which determine whether the load on the least-loaded cluster could be redistributed to the other clusters without spinning up the cluster again.
Economy	Conserves credits by favoring keeping running clusters fully-loaded rather than starting additional clusters, which may result in queries being queued and taking longer to complete.	Only if the system estimates there's enough query load to keep the cluster busy for at least 6 minutes.	After 5 to 6 consecutive successful checks (performed at 1 minute intervals), which determine whether the load on the least-loaded cluster could be redistributed to the other clusters without spinning up the cluster again.

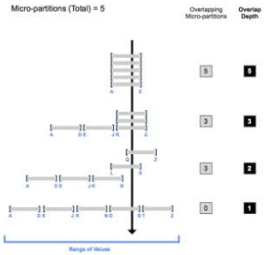
- Some rules:

- Maximized:
    - ↑ **max & min:** Specified number of clusters start immediately.
    - ↓ **max & min:** Specified number of clusters shut down when they finish executing statements and the auto-suspend period elapses.
- Auto-scale:
    - ↑ **max:** If `new_max_clusters > running_clusters`, no changes until additional clusters are needed.
    - ↓ **max:** If `new_max_clusters < running_clusters`, excess clusters shut down when they finish executing statements and the scaling policy conditions are met.
    - ↑ **min:** If `new_min_clusters > running_clusters`, additional clusters immediately started to meet the minimum.
    - ↓ **min:** If `new_min_clusters < running_clusters`, excess clusters shut down when they finish executing statements and the scaling policy conditions are met.

// Tables in Snowflake //

Micro-partition

- All of SF Tables are divided into micro-partition
- Each micro-partitions are compressed columnar data
  - Max size of 16mb compressed (50-500MB of uncompressed data)
  - Stored on logical hard-drive
  - Data only accessible via query through SF not directly
  - They are immutable, can't be changed
- Order of data ingestion are used to partition these data
- SF uses natural clustering to colocate column with the same value or similar range
  - Results to non-overlapping micro-partition and least depth
    - Improves query performance to avoid scanning unnecessarily micr-partitions



- Metadata of the micropartitions are stored in the Cloud Services Layer
  - Customers can query this without needing an Active vwh
    - Range of each columns
    - Number of Distinct Values
    - Count NULL

>Clustering Key

- SF recommends the use of Clustering key once your table grows really large (multi-terabyte range).
  - Especially when data does not cluster optimally.
- Clustering keys are subset of columns designed to colocate data in the table in the same micro-partitions
  - Improves query performance on large tables by skipping data that is not part of filtering predicate

Zero-copy Cloning

- Allows Customers to Clone their table
- SF references the original data (micro-partition) - hence zero-copy (no additional storage cost)
  - When a change is made to the cloned table (new data added, deleted or updated) then a new micro-partition is created (incurs storage cost).
- Cloned object do not inherit the source's granted privileges

Types of Tables (Internal)

>Temporary Tables

- Used to store data temporarily
- Non-permanent and exists only within the session - data is purged after session ends
- Not visible to other users and not recoverable
- Contributes to overall storage
- Belongs to DB and Schema - Can have the same name as another non-temp table within the same DB and Schema
- Default 1 day Time Travel

>Transient Tables (or DB and Schema)

- Persists until dropped
- Have all functionality as permanent table but no Fail-safe mode (no FS storage cost)
- Default 1 day Time Travel

>Permanent Table

- Have 7 days fail safe
- Default 1 day Time Travel (up to 90 days for >=Enterprise edition)

>External Table

- Allows access to data stored in External Stage

>Creating tables

- SF doesn't enforce any constraints (primary key, unique, foreign key) beside the NOT NULL constraint

Types of Views

>Non-materialized views (views)

- Named definition of query
- Result are not stored

>Materialized views

- Result are stored
- Faster performance than normal views
- Contributes towards storage cost

	Performance Benefits	Security Benefits	Simplifies Query Logic	Supports Clustering	Uses Storage	Uses Credits for Maintenance	Notes
Regular table				✔	✔		
Regular view		✔	✔				
Cached query result	✔						Used only if data has not changed and if query only uses deterministic functions (e.g. not CURRENT_DATE).
Materialized view	✔	✔	✔	✔	✔	✔	Storage and maintenance requirements typically result in increased costs.

>Secure view

- Both non-materialized and materialized view can be defined as Secure view
- Improves data privacy - hides view definition and details from unauthorised viewers
- Performance is impacted as internal optimization is bypassed

// Time Travel and Fail Safe //

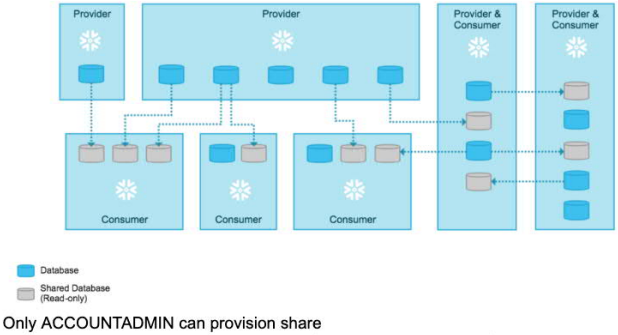
Introduction

>Time Travel (0-90 days)
- Allows db to query, clone or restore historical data to tables, schema or db for up to 90 days
  - Can use this to access snapshot of data at a point in time
- Useful if data was deleted, dropped or updated.

>Fail safe (7 days)
- Allows disaster recovery of historical data - Only accessible by Snowflake!

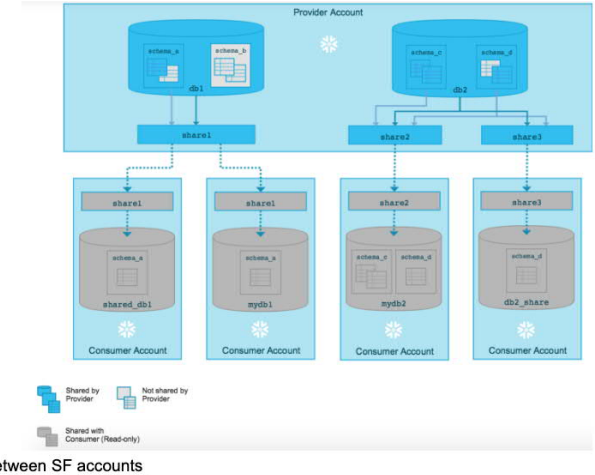
// Data Sharing //

Introduction



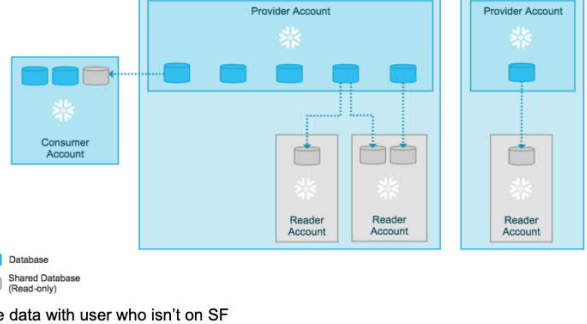
- Only ACCOUNTADMIN can provision share
- The sharer is the Provider while the user of the shared data is the Consumer
- Only ACCOUNTADMIN role can create this
- Secure data sharing enables account-to-account sharing of Database, Tables and Views
  - No actual data is copied or transferred
- Sharing accomplished using SF service layer and metadata store
- Charge is compute resources used to query shared data
- User creates share of DB then grants object level access
  - read-only db is created on consumer side
- Access can be revoked at any time
- Type of share includes Share and Reader
- No limit on number of shares or accounts but 1 DB per share

Share



- Between SF accounts
- Each share consists of
  - Privilege that grant db access
  - Privilege that grant access to specific object (tables, views)
  - Consumer accounts with which db and object are shared
- Can perform DML operations

Reader Account



- Share data with user who isn't on SF
- Objects can only be read not modified (no DML) - consumer can only consume the data

// Access Control within Snowflake //

Network Policy
- Allows access based on IP whitelist or restrictions to IP blacklist
    - Apply through SQL or WebUI
    - Only ACCOUNTADMIN or SECURITYADMIN can modify, drop or create these

PrivateLink
- If time permits briefly read through these (AWS and Azure)

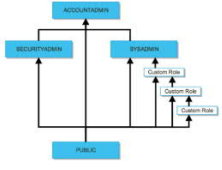
MFA
- Powered by Duo System and enrolled on all accounts, Customers need to enable it
- Recommend enabling MFA on ACCOUNTADMIN
- Use with WebUI, SnowSQL, ODBC, JDCB, Python Connector

Federated Auth and SSO
- User authenticate through external SAML 2.0-compliant identity provider (IdP)
    - Users don't have to log into Snowflake directly
- As per basic introduction to SSO, a token is passed to the application that the user is trying to login to authenticate which in turns will open access when verified

Access Control Models

SF approach to access control uses the following models:
- Discretionary Access Control (DAC) - All object have an owner, owner can grant access to their objects
- Role-based Access Control (RBAC) - Privileges are assigned to roles, then roles to users
    - Roles are entities that contains granted privileges (level of access to object) which are

System Defined Roles



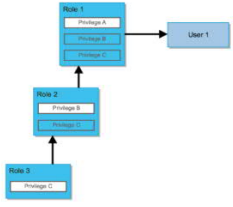
>ACCOUNTADMIN
- encapsulates SECURITYADMIN and SYSADMIN

>SECURITYADMIN
- Creates, modify and drops user, roles, networks, monitor or grants

>SYSADMIN
- Has privileges to create vwh, and db and its objects
- Recommend assigning all custom roles to SYSADMIN so it has control over all objects created (create role hierarchy based on this)

>PUBLIC
- Automatically grant to all users
- Can own secured objects
- Used where explicit access control is not required

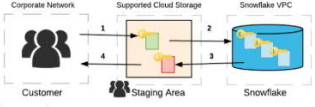
>Custom Roles
- Created by SECURITYADMIN
- Assigned to SYSADMIN
    - If not assigned, only roles with MANAGE GRANTS can modify grants on it.
- Create custom roles with least privilege and role them up



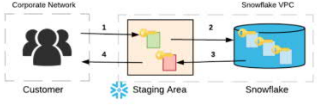
- Visit "Access Control Privileges" in SF Doc and have a quick skim over the privileges

// Data Security within Snowflake //

> End-to-end Encryption
- Snowflake encrypts all data by default, no additional cost
- No one other than the customer or runtime components can read the data
    - Data is always protected as it is in an encrypted state
- Customer provides encrypted data to the external staging area (S3 etc)
    - Provide SF with the encryption master key when creating the Named Stage



    - Client Side Encryption
- Customer provides unencrypted data to SF internal staging area, SF will automatically encrypt the data.



>Key Rotation
- Snowflake encrypts all data by default and keys are rotated regularly
- Uses Hierarchy Key Model for its Encryption Key Management - comprises of
    - Root Key
    - Account Master Key (auto-rotate if >30 days old)
    - Table Master Key (auto-rotate if >30 days old)
    - File Keys

>Tri-Secret Secure and Customer-managed keys (Business Critical Edition)
- Combines the customer key with snowflake's maintained key
    - Creates composite master key then use it to encrypts all data in the account
    - If customer key or snowflake key is revoked, data cannot be decrypted