Snowflake fundamentals

servian_

What is Snowflake

Snowflake is a modern data platform built for any cloud.

It is an analytic data warehouse provided as Software-as-a-Service (SaaS). Snowflake provides a data warehouse that is faster, easier to use, and far more flexible than traditional data warehouse offerings.

Snowflake's data warehouse is not built on an existing database or "big data" software platform such as Hadoop.

The Snowflake data warehouse uses a new SQL database engine with a unique architecture designed for the cloud. To the user, Snowflake has many similarities to other enterprise data warehouses, but also has additional functionality and unique capabilities.

A NEW ARCHITECTURE FOR DATA WAREHOUSING

Traditional Architectures

Shared-disk

Shared storage
Single cluster

Shared-nothing

Decentralized, local storage
Single cluster

Snowflake

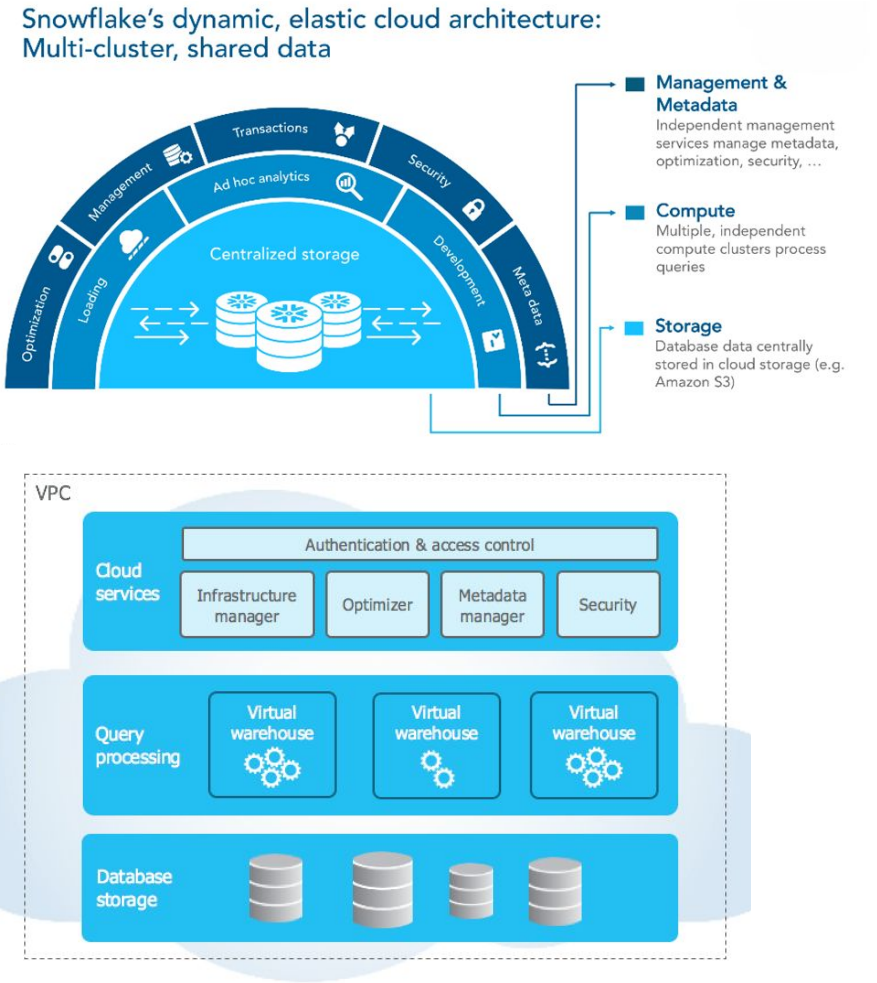
Multi-cluster, shared data

Centralized, scale-out storage
Multiple, independent compute clusters

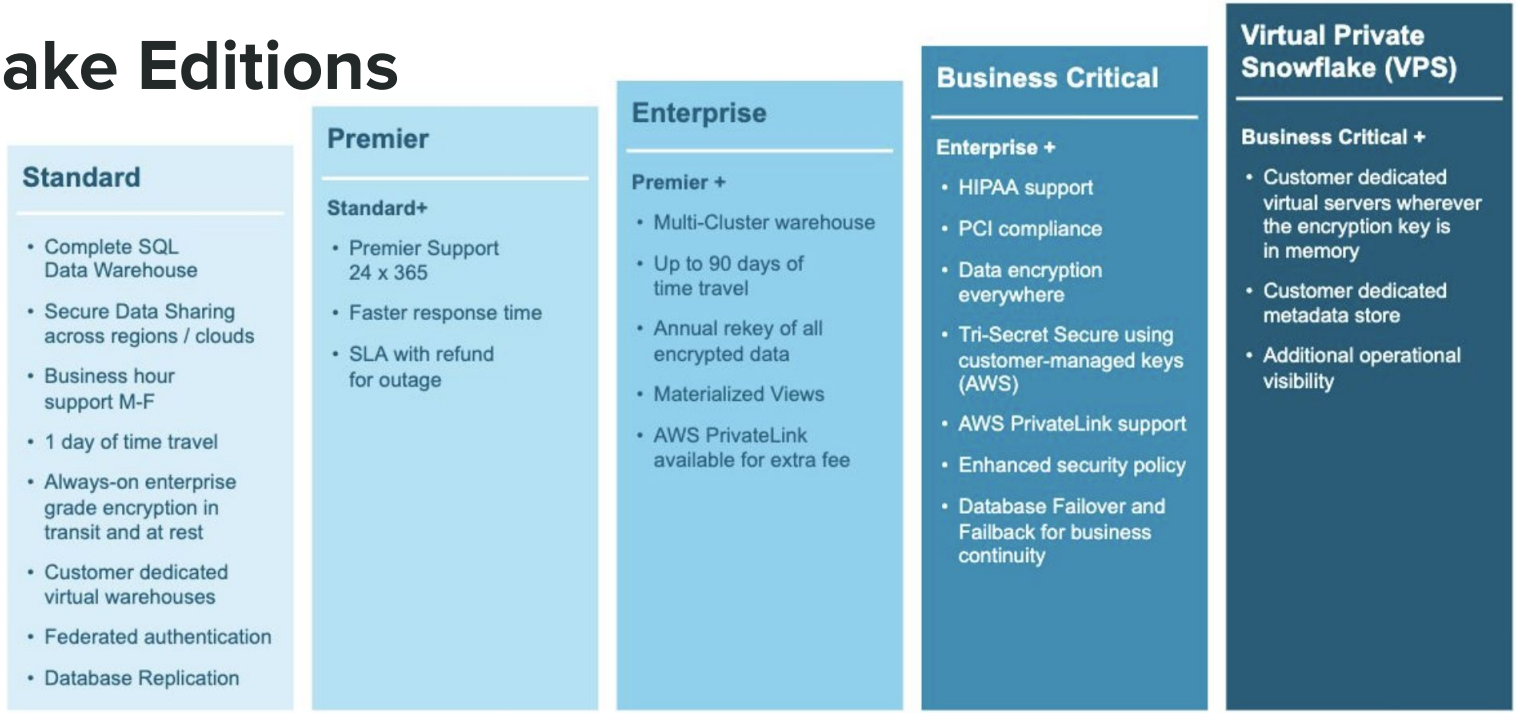
Snowflake's Architecture

Snowflake's architecture is

- a hybrid of traditional shared-disk and shared-nothing database architectures.
- Snowflake uses a central data repository for persisted data that is accessible from all compute nodes in the platform.
- Snowflake processes queries using MPP (massively parallel processing) compute clusters where each node in the cluster stores a portion of the entire data set locally.
- Snowflake provides the performance and scale-out benefits of a shared-nothing architecture.

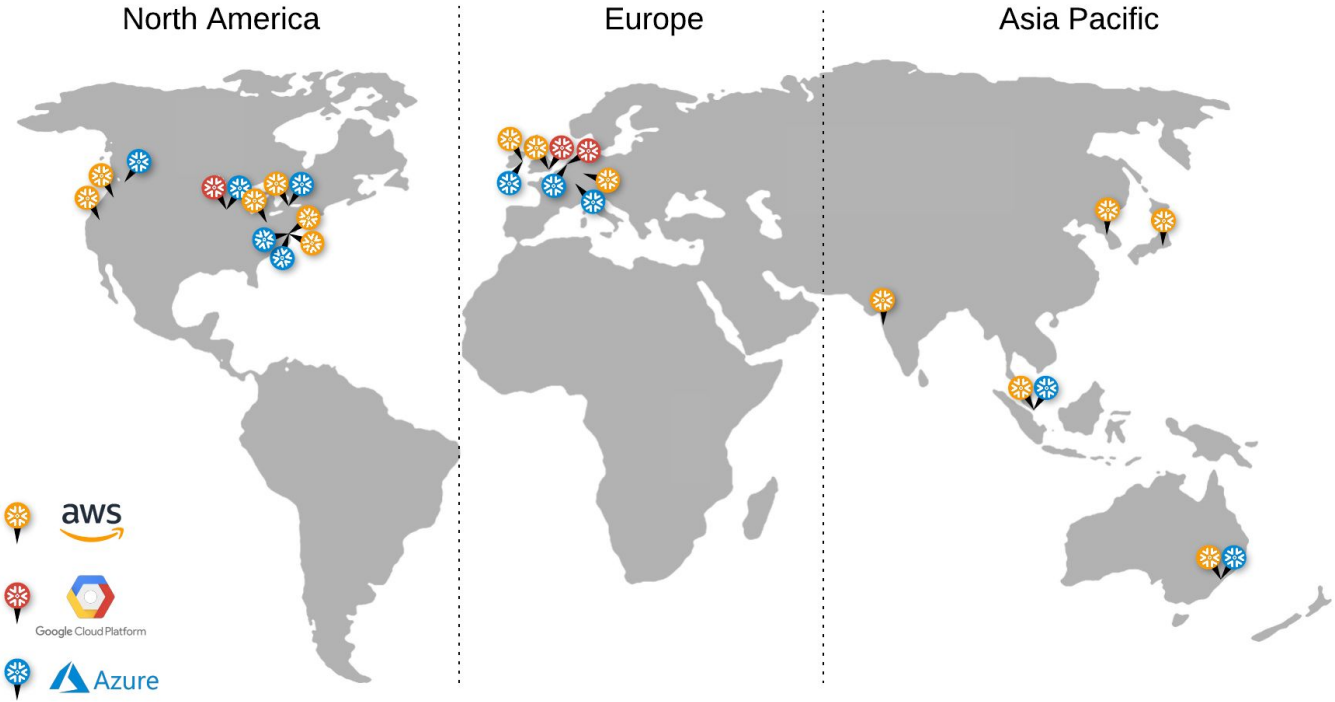


Snowflake Editions



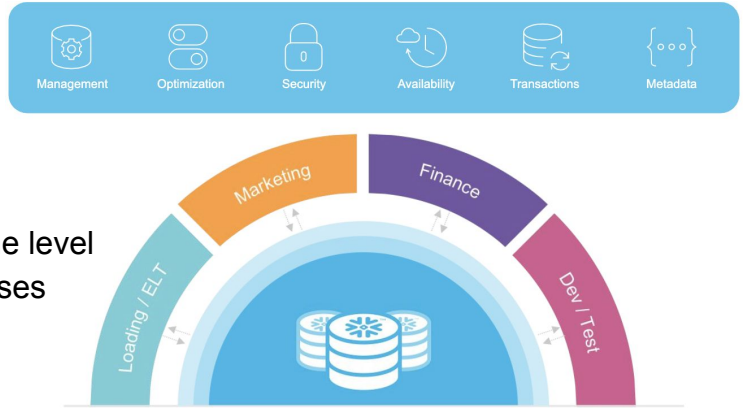
- Replication is physical - eg for failover; not like cloning
- Soon can failover to another cloud vendor in another region
- Currently as DB level; soon at account level
- Storage is at cloud vendor's price; cheaper with capacity plan
- Cost per credit = 1 hour @ x-small; minimum 1 minute, then per second

Clouds and locations



Snowflake architecture: Cloud Services

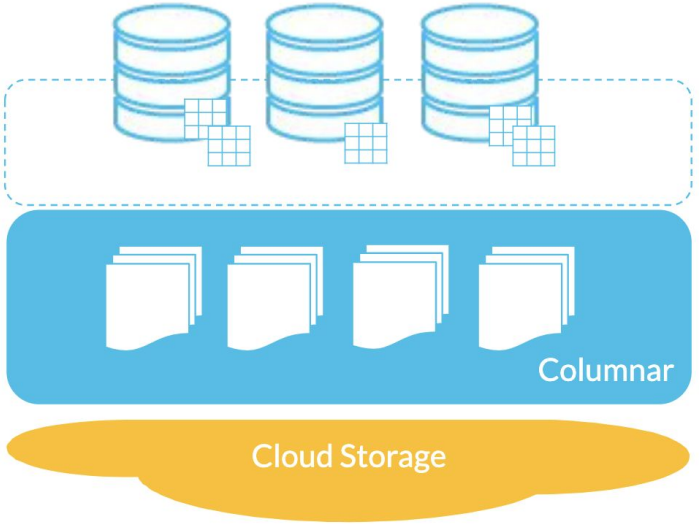
- Brains of the service
- Coordinates activities
- Runs on compute instances provisioned by Snowflake
- Built in continuous availability
  - Synchronises across availability zones
  - Fully online updates and patches - everyone is on the same level
  - In future, can choose to be a day ahead or behind of releases
- SQL Optimiser
  - Cost-based
  - Can now use explain plan
  - Auto join order optimisation
  - Auto stats gathering
  - Pruning using metadata about micro-partitions
  - No hints (yet)
- Transaction service
  - CRUD, ACID, Concurrency



- Security
  - Authentication
  - Access control for users and roles
  - Access control for shares
  - Encryption and key management
- Metadata management

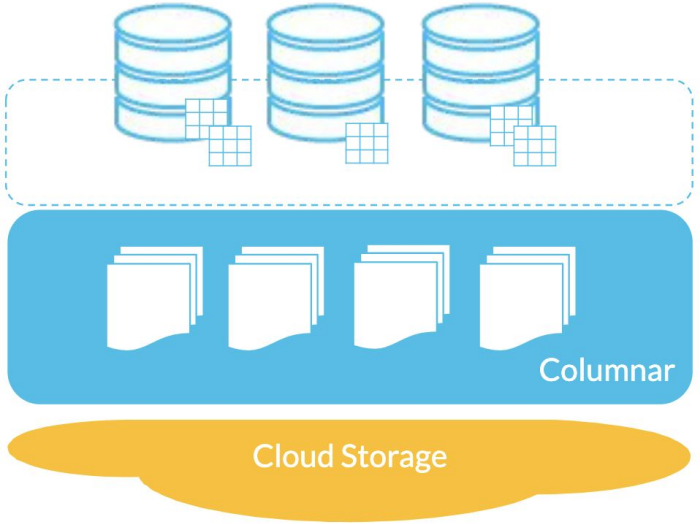
Data storage layer

- Hybrid Columnar => can be more than 1 column in a micro partition
- Automatic micro-partitioning
- Natural data clustering and optimization
    - Usually clustering makes sense for tables > 1Tb
- Native Semi-Structured data support and optimization
- All storage within Snowflake is billable (compressed)
- Micro-partitions
    - contiguous units of storage ~ 3-30x compressed
    - max 16Mb (generally)
    - many per table
    - immutable
    - services layer stores metadata
        - min & max (range of values in each column)
        - number of distinct values



Micro-partitions

- Micro-partitions
  - contiguous units of storage ~ 3-30x compressed
  - max 16Mb (generally)
  - many per table
  - immutable
  - services layer stores metadata
    - min & max (range of values in each column)
    - number of distinct values
- Physical data files that comprise Snowflake's logical tables
- Automatically-created contiguous units of storage, partitioned based on ingestion order
- Attempts to preserve natural data co-location
- Immutable - updates create new micro-partition versions



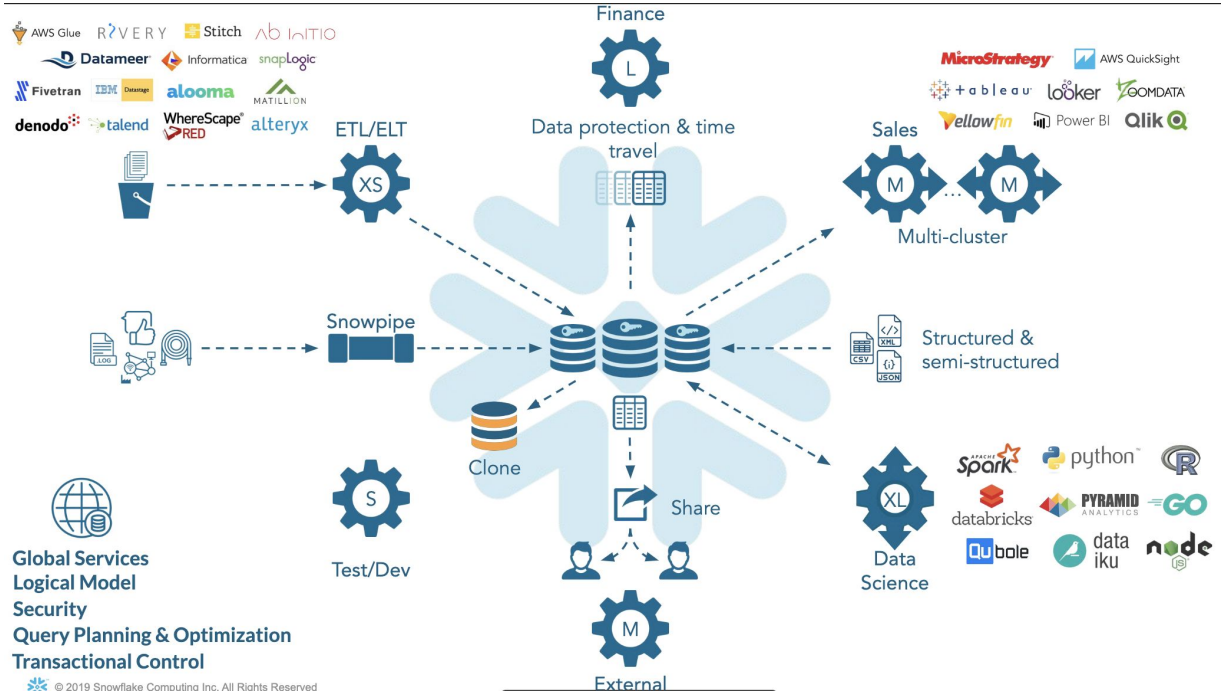
Snowflake Virtual Warehouse

A virtual warehouse, often referred to simply as a “warehouse”, is a cluster of compute resources in Snowflake. A warehouse provides the required resources, such as CPU, memory, and temporary storage, to perform operations in a Snowflake session.

Warehouses are required for queries, DML operations, including loading data into tables. A warehouse is defined by its size(t-shirt sizing), as well as the other properties that can be set to help control and automate warehouse activity.

Warehouses can be started and stopped at any time. They can also be resized at any time, even while running, to accommodate the need for more or less compute resources, based on the type of operations being performed by the warehouse.

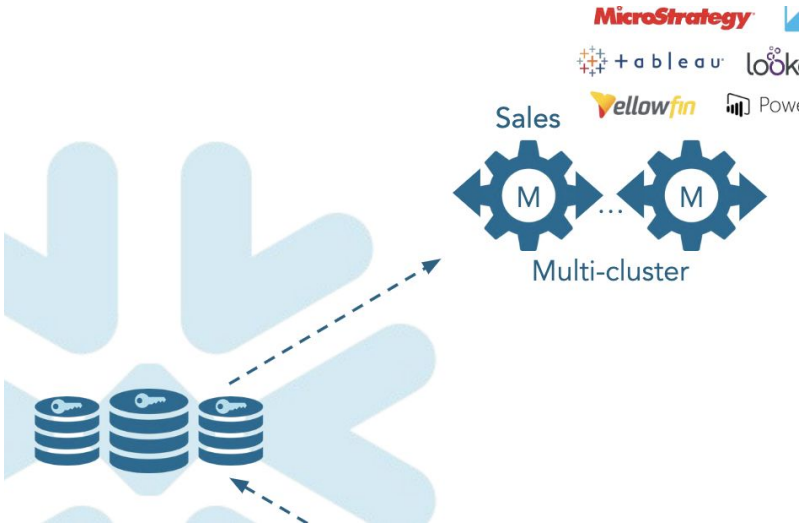
Virtual Warehouses



Each has its own resources (and sizing)
Size can be changed instantly with SQL - ALTER WAREHOUSE RESIZE
Share the data
Snowpipe - for streaming, trickle-feeds - near real time, eg 1 minute; doesn't need resource
D Sc - Data Robot connector

MULTI-CLUSTER

- Up to 10 per cluster
- Auto-scale or manual
- **Scale up for performance; scale out for concurrency**
- Monitor queuing time on a vwh - use clustering to reduce
- For test/dev - with zero-copy cloning
- TIme-travel
  - old version of data is kept for up to 90 days (can specify)
- Data sharing
  - Live data
  - Using the client's compute resources
  - Data Exchange is currently free



Compute - Virtual warehouses

- A named wrapper around a cluster of servers with CPU, memory, and SSD in the cloud
- The larger the warehouse, the more servers in the cluster, the more resources it has
- Extra Small (XS) has one server per cluster
    - Each size up doubles in size
- Jobs requiring compute run on virtual warehouses
- While running, a virtual warehouse consumes Snowflake credits
    - You are charged for compute
- Should reflect units of workload management
    - ELT
    - BI
    - D Sc

- Standard
    - Will only ever have a single Compute luster
    - Cannot “scale out
- Multi-Cluster Warehouse (MCW)
    - Can spawn additional Compute Clusters (scale out) to manage changes in user and concurrency needs
    - Enterprise Edition feature

Virtual warehouse sizing and credits

- Sizes
  - Warehouses are sized in “t-shirt” sizing
  - Size determines the number of servers that comprise each cluster in a warehouse
  - Each larger size is double the preceding, in both VMs in the cluster and in Snowflake credits consumed
- Credits
  - Credits are how you are billed for compute usage
    - You may have a set number of credits
    - You may be billed monthly for your credits
  - Credits are charged based on the number of virtual warehouses you use, their size, and how long you use them
- Warehouse usage (or compute) is charged per-second, with a one-minute minimum
- It’s easier to tell when you need a bigger size than when you could make do with a smaller one
- Resizing a warehouse
  - Can be completed at any time, even when running
  - Completed via ALTER WAREHOUSE statement or the UI
  - Effects of resizing:
    - Suspended Warehouse: will start at new size upon next resume
    - Running Warehouse: immediate impact; running queries complete at current size, while queued queries run at new size

Scale up and out

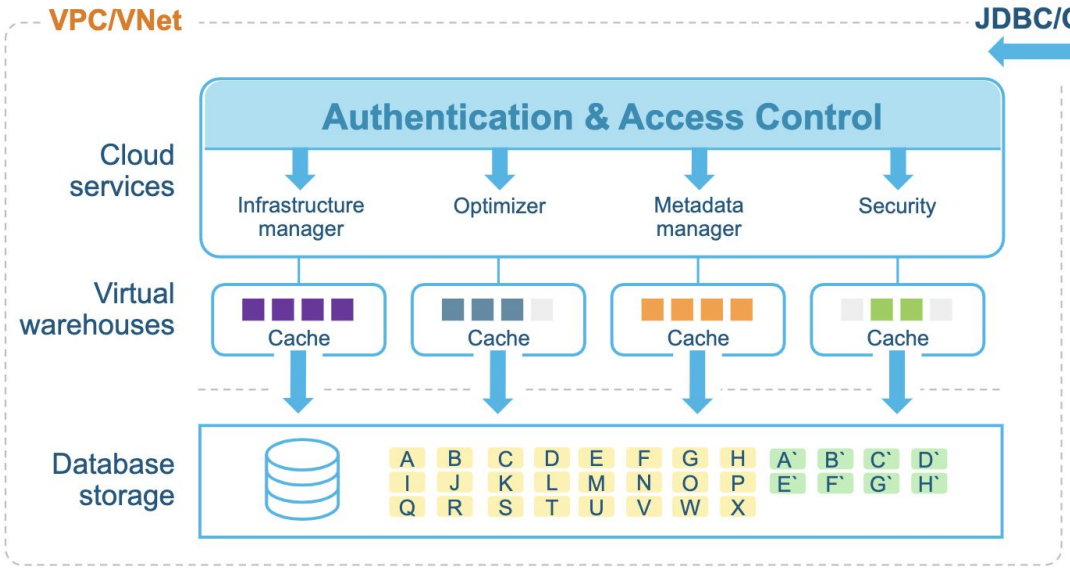
Scale up for performance

- Elastic Processing Power (CPU, RAM, SSD)
  - Raw performance boost for complex queries or ingesting large data sets
  - Typically more complex queries on larger datasets require larger Warehouses
  - Not intended for handling concurrency issues (more users/queries)
- Warehouse Sizing Guidelines
  - Snowflake uses per-second billing: use larger warehouses for more complex workloads and (auto) suspend when not in use
  - Keep queries of similar size and complexity on the same warehouse to simplify compute resource sizing

Scale out for concurrency

- General Functionality and Considerations
- Single Virtual Warehouse with multiple compute clusters
- Delivers consistent SLA, automatically adding and removing compute clusters based on concurrent usage
- Scale out during peak times and scale back during slow times
- Queries are load balanced across the clusters in a Virtual Warehouse
- Deployed across availability zones for high availability
- Guidelines
  - MIN_CLUSTER_COUNT : 1-10 (default 1)
  - MAX_CLUSTER_COUNT: 1-10 (default 1) >= MIN_CLUSTER_COUNT

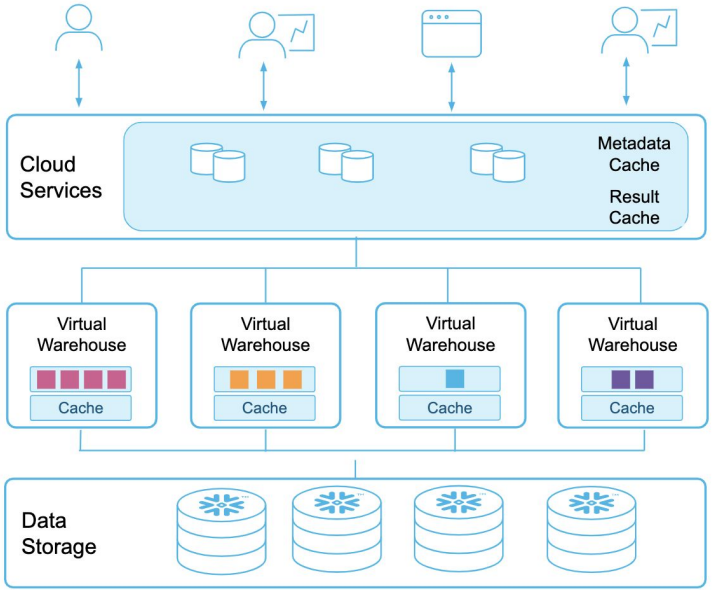
MULTI-CLUSTER SHARED DATA ARCHITECTURE



- Storage decoupled from compute
- All data in one place
- Dynamically combine storage and compute

Snowflake is physically installed with a huge set of resources in each region/location per cloud vendor

Caching in Snowflake



METADATA CACHE

QUERY RESULT CACHE

COMPUTE DATA CACHE

Query results cache - kep for 24 hours

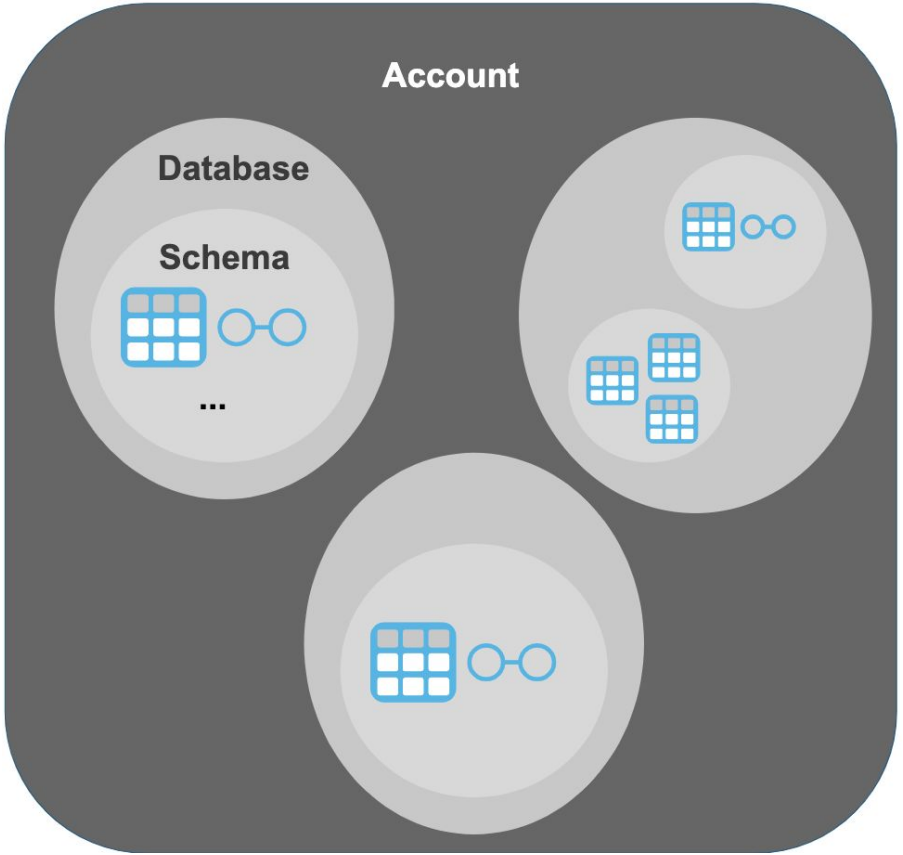
3 layers of caching

Snowflake structure

Local data organisation

Databases and schemas logically organize data within a Snowflake account

- A database is a logical grouping of schemas
- Each database belongs to a single account
- A Schema is a logical grouping of database objects, such as tables and views



Snowflake objects

- Roles own objects; not users
- Can share database with other accounts (read only)
    - Use warehouses to split costs across departments
    - Federated master data model would need custom config to share changes back and ingest
-

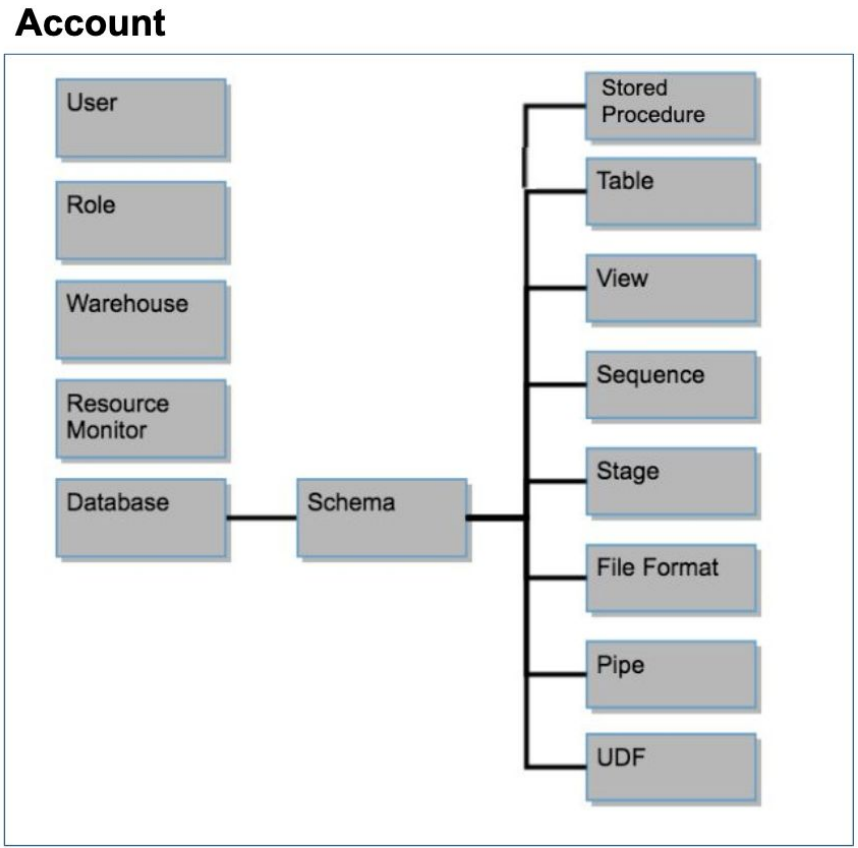


Table types

Permanent

- Persist until dropped
- Default
- Up to 90d time travel
  - only for permanent tables
  - magic happens at micro-partition level
- Fail safe for 7 days after time travel - not intended for normal operational access, but there for emergencies
- Consider storage costs for time travel and fail safe!

Transient

- Persist until dropped
- Multi user
- For data that doesn't need data retention like permanent
- Saves cost
- Multiple users
- Good for data science

External

- Persist until removed
- Snowflake over external data lake
- Data accessed via an external stage
- Read-only
- Lots of work going on to make it faster
- Can define partitions - so Snowflake knows which data is in which files

Temporary

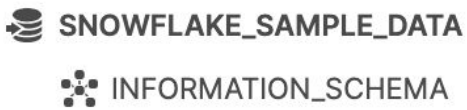
- Persist for session
- eg for ELT
- Single user

View types

Standard	Secure	Materialized
Default	Definition and details visible to authorised users	More like a table - stored results of underlying query
Named definition of an SQL statement	Executes as owning role	Auto-refreshed
Executes as executing role		Secure MV is also supported
Anyone can view DDL	Must use when sharing data externally	Automatically makes sure data is consistent, etc
		Currently limited to only 1 source table
		Great for aggregations and filtering a commonly used subset of a table

Using the Snowflake Web UI

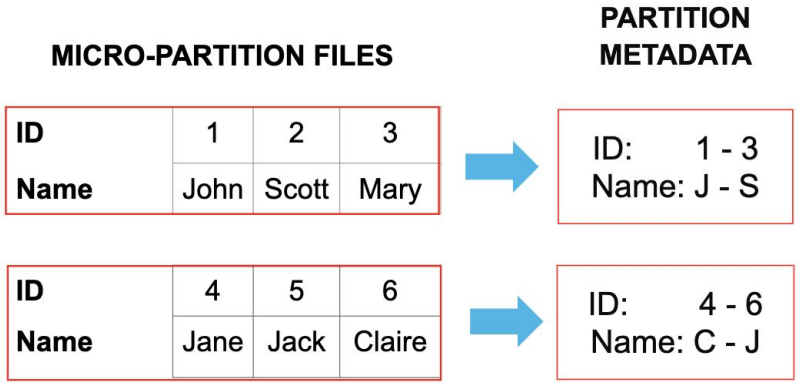
- Dialogs have show-sql so you can harvest SQL
- Every DB has information schema and public
- Shares - need admin priv or similar to set these
  - inbound and outbound
- Worksheets is the most-used thing
  - Worksheets save automatically
- Context = role + warehouse + database + schema
  - USE ROLE myRole
- Auto highlight on/off - need semicolons at statement ends in order to run together
- Snowflake has bought a company so they can improve the web UI
- Arrow on the DB icon means it has been shared from outside



- Remember that warehouses are compute resources - not data
  - set auto-suspend and auto-resume to save resource costs
  - Also manipulated with SQL - unlike traditional databases
- Reader accounts - for people who don't have a Snowflake account
- Third-party product trials - Stitch, Fivetran, Matillion

Metadata

- Snowflake automatically collects and maintains metadata about tables and their underlying micro-partitions, including:
- Table level
  - Row count
  - Table size (in bytes)
  - File references and table versions
- Micro-partition column level:
  - Range of values
  - Number of distinct values
  - MIN and MAX values
  - NULL count



Create and manage tables

- Generally ANSI DDL, except:
  - CLONE - good for snapshots; table, schema or database
  - UNDROP
  - SHARE
  - CREATE TRANSIENT TABLE
  - CREATE TABLE .. LIKE - copy DDL without data
  - STAGE
    - If you don’t specify location URL during CREATE, Snowflake will create it within its own storage
  - FILE FORMAT
  - PIPE - a wrapper for the COPY command
- Clone creates a new set of metadata
  - No immediate changes to micro-partitions
  - Micro-partitions are owned by original object’s owner
- CREATE TABLE My_Table -> MY_TABLE; “My_Table” -> My_Table
- Views - ANSI except:
  - SECURE VIEW
  - CLUSTER BY
- Sequences exist

Using Snowflake

- Partner Connect
- BI tools most used - Tableau, Power BI, Looker, Mode, ..
    - Tableau has free dashboards to show usage stats
- ETL/ELT most used - Fivetran (not in AU), Matillion, Stitch, Talend, Segment, Informatica, Alooma
    - Matillion has better pay-per-use model than Stitch
    - Informatica takes data to their cloud (security, cost, time)
    - Fivetran is used by Snowflake to populate stats for our reporting
- Analytics - Spark, Databricks, R, Alteryx, SAS
- Connecting Clients - Snowflake UI, JDBC, Python, ODBC, SnowSQL

DML

- MERGE
- File commands
  - PUT <to a stage> (need snowsql to do whole folders) - from local into Snowflake-provided bucket
    - PUT only works for Snowflake bucket; have to use cloud providers CLI etc for theirs
    - File size for best batch performance ~ 100Mb compressed for using lot of warehouses concurrently
    - File size for streaming - small
  - GET, LIST, REMOVE
- PIVOT .. IN - PIVOT (SUM(sales) FOR month) IN ('JAN','FEB','MAR')
- TABLE function - treat function results as table
  - SELECT * FROM TABLE(INFORMATION_SCHEMA.login_history_by_user());
- Dates - DATE and TIMESTAMP datatypes
  - DATEADD like SQL Server
- LIMIT - like SQL Server and mySQL
  - SELECT TOP *n*
- CROSS JOIN - no join conditions - like SQL Server
- NATURAL JOIN - only returns one set of columns for the join column(s)

COPY INTO and VALIDATE

- Load/unload commands
  - COPY INTO <table> | <location>
    - Pipe is a wrapper for this
  - VALIDATE
- Example
  - PUT file:///tmp/load/contacts*.csv @my_stage AUTO_COMPRESS=true;
    - AUTO_COMPRESS defaults to true
  - COPY INTO mytable
    - FROM @my_stage/contacts1.csv.gz
    - FILE_FORMAT = (format_name = mycsvformat)
    - ON_ERROR = 'skip_file';
- COPY INTO my_table
  FROM @my_stage
  FILE_FORMAT= (FORMAT_NAME = mycsvformat)
  pattern='.*contacts[1-5].csv.gz'
  ON_ERROR = 'skip_file';
- CREATE TABLE save_copy_errors AS
  SELECT * FROM TABLE(VALIDATE(my_table,job_id=>'<query_id>>'));

Collation

- Applies to data in STRING columns
- Can now specify for account, database, schema
- in CREATE command - affects stats gathering but not storage (we think)
- Default is UTF8
- Options include:
    - local language
    - case sensitivity
    - case conversion
    - space trimming
- SELECT .. COLLATE ‘en-ci’ => case-insensitive; ‘en-ci-pi’ also punctuation-insensitive

Query profile

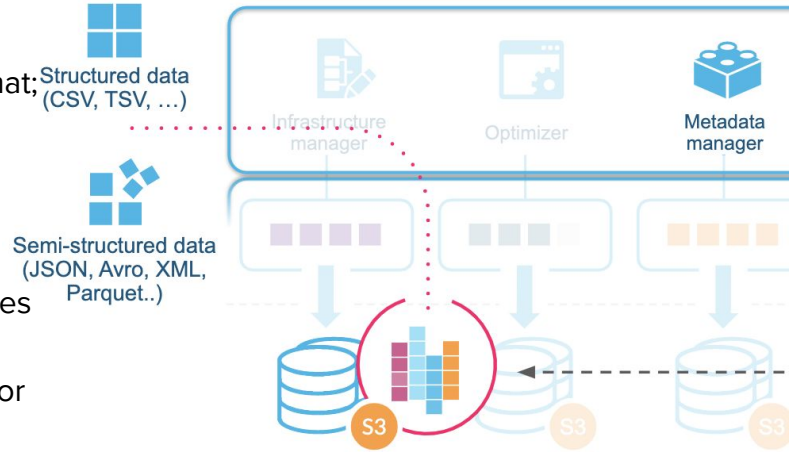
- Easiest to use GUI, but can use programmatically
- History tab
  - local storage = WH cache - ie the SSD on that compute node
- Profile tab
  - types of results depends on the nature of the query; eg metadata-only will be simple
  - security settings can limit how much you can see
- Node execution time
  - Initialization = setup activities prior to processing
  - Processing = CPU data processing
  - Local Disk I/O = blocked on (read from/write to) local SSD on node
  - Remote Disk I/O = blocked on (read from/write to) remote cloud storage
  - Network Communication = blocked on network data transfer (read from/write to)
  - Synchronization = various sync activities between processes
- Note - an always-on WH doesn't keep losing its cache
- Things to do about big, slow tables
  - Add cluster keys - can be expensive; physically reorgs
  - Materialised view
  - Create table as select

Statistics

- Scan progress = % of data scanned for table thus far
- Bytes scanned = local + remote I/O
- Percentage scanned from cache = local / (local + remote)
- Bytes written = bytes into Snowflake (table)
- Bytes written to result = bytes to result object
- Bytes read from result = bytes from result object
- External bytes scanned = bytes from external object (stage)
- Bytes sent over the network = peer-peer data exchange
- Partitions scanned = number of micro-partitions read (local + remote)
- Partitions total = number of micro-partitions in table
- **Bytes spilled to local storage = written to local SSD on node (insufficient memory)**
- **Bytes spilled to remote storage = written to remote cloud storage (insufficient local SSD)**

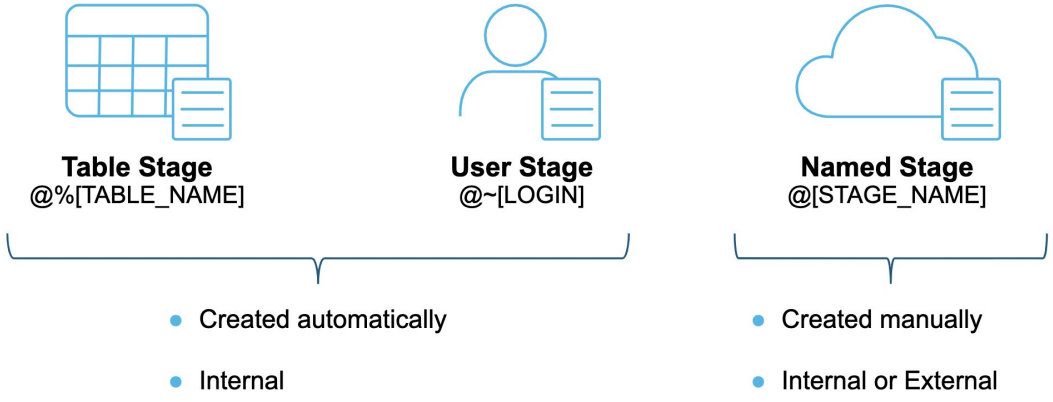
Data loading steps

- Outbound - only Parquet, JSON, csv
- Workflow
  - CREATE FILE FORMAT
  - CREATE TABLE
  - COPY INTO TABLE
- Local storage example:
  - CREATE STAGE my_stage FILE_FORMAT = my_csv_format;
  - PUT FILE:///data/data.csv @my_stage;
  - COPY INTO my_table FROM @my_stage;
- Note: Compression during the PUT uses local resources. The local host needs sufficient memory, and space in /tmp, or the PUT will fail.



Metadata created: Statistics for databases, tables, columns, & files calculated & stored in cloud services layer

Stages



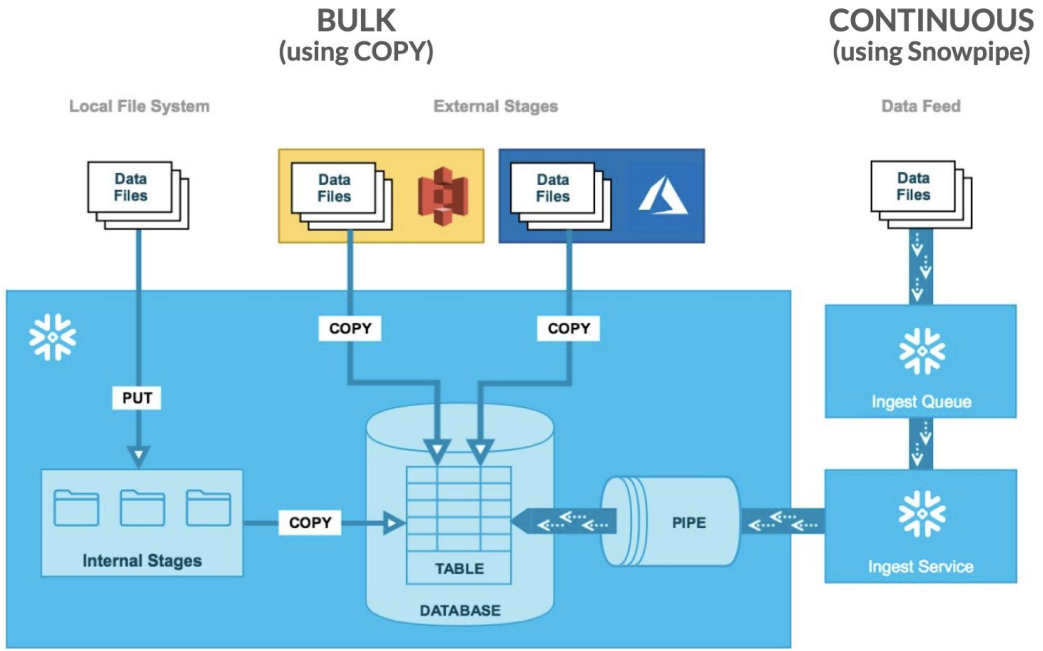
- Cloud file repository
- Simplifies and streamlines bulk loading and unloading
- Can be internal or external
  - Internal: stored internally in Snowflake
  - External: stored in an external location
- Best Practice: Create stage object to manage ingestion workloadAutomatically created stages:
  - Not really needed by users
- Named stages:
  - Can transfer owning role
- CREATE FILE FORMAT DEMO_FF TYPE = 'CSV' FIELD_DELIMITER = ',' RECORD_DELIMITER = '\n' SKIP_HEADER = 1;
- Can bind a file format to a stage, so the stage always knows what format to expect

File formats and loading from stages

- CREATE FILE FORMAT DEMO_FF TYPE = 'CSV' FIELD_DELIMITER = ',' RECORD_DELIMITER = '\n' SKIP_HEADER = 1;
- Can bind a file format to a stage, so the stage always knows what format to expect
- csv is more flexible - detect format from header
- Cloud storage example:
    - CREATE STAGE my_s3_stage
      url='s3://mybucket/encrypted_files/'
      CREDENTIALS=(aws_key_id='******'
      aws_secret_key='******')
      ENCRYPTION=(master_key = '********')
      FILE_FORMAT = my_csv_format;
    - COPY INTO my_table FROM @my_s3_stage PATTERN='.*sales.*.csv';
- Data can be copied directly from a Stage or using a SELECT statement

Bulk loading versus continuous

- Streaming - can listen to a SQ queue ~ 1 minute micro-batches
- Don’t use Snowpipe for big bulk stuff - you’ll get a please explain



Data loading recommendations

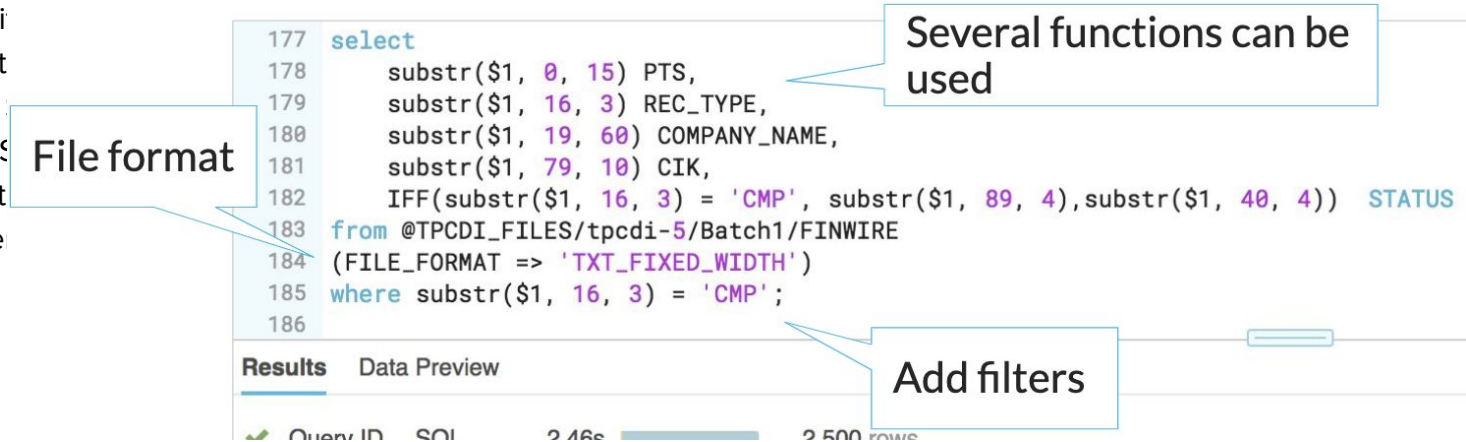
- File size and number
  - Split large files before loading into Snowflake
    - So they can load in parallel and use more compute
  - 10 - 100Mb
  - About 8 files in parallel on XS - double for S, etc ..
- File locations
  - Many locations with fewer files is faster - scanning few files by targeting directories
  - Organize data in logical paths (e.g., subject area and create date) /system/market/daily/2018/09/05/
  - Use wildcards late in the file path definition, to reduce scanning:
    - COPY INTO table FROM /system/market/daily/2018/*
- Purge - works on Snowflake/internal stages; not sure about external

Speed, concurrency, locking

- 10-20% faster to load structured than VARIANT
- csv loads 2-3 times faster than parquet, etc
- COPY and INSERT typically do not acquire a partition lock.
    - They acquire table lock for a brief time during transaction commit.
- UPDATE, DELETE and MERGE acquire partition lock, hence:
    - Use staging tables to manage MERGE
    - Consolidate UPDATEs and DELETEs when possible
- Fully ACID compliant

COPY INTO versus INSERT

- Only basic transformation in COPY
  - COPY INTO home_sales (city, zip, sale_date, price)
    FROM (SELECT SUBSTR(t.$2,4), t.$1, t.$5, t.$4 FROM @my_stage t)
    FILE_FORMAT = (FORMAT_NAME = MYCSVFORMAT);
  - COPY INTO casttb(col1, col2, col3)
    FROM (SELECT TO_BINARY(t.$1, 'utf-8'), TO_DECIMAL (t.$2, '99.9', 9, 5), TO_TIMESTAMP_NTZ(t.$3)
    FROM @~/datafile.csv.gz t) FILE_FORMAT = (TYPE = CSV)
- Snowflake is optimized for bulk load and batched DML using the COPY INTO command
- Use COPY INTO to load data rather than INSERT with SELECT
- Use INSERT only i
- Batch INSERT stat
  - INSERT w/
- CREATE TABLE AS
- Minimize frequent
- Better to use exte



Data load management

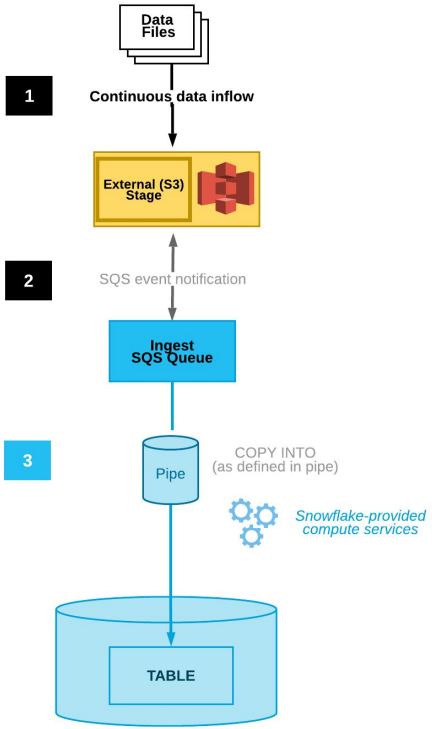
- Error handling - ON_ERROR
- Validate before load
  - COPY INTO my_table FROM @my_stage/mylife.csv.gz VALIDATION_MODE=return_all_errors;
  - SET qid=LAST_QUERY_ID();
  - COPY INTO@my_stage/errors/load_errors.txt FROM (SELECT rejected_record FROM TABLE(result_scan($qid))));
- Validate after load
  - Validates the files loaded in a past execution of the COPY INTO and returns all errors encountered during the load, rather than just the first error
  - SELECT * FROM table(VALIDATE(mytable, job_id => '<query_id>'));
  -
- Monitoring
  - History tab of the UI
  - SELECT * FROM information_schema.load_history
  - Kept for 2 weeks; longer term available somewhere else we'll learn about later

Pipes

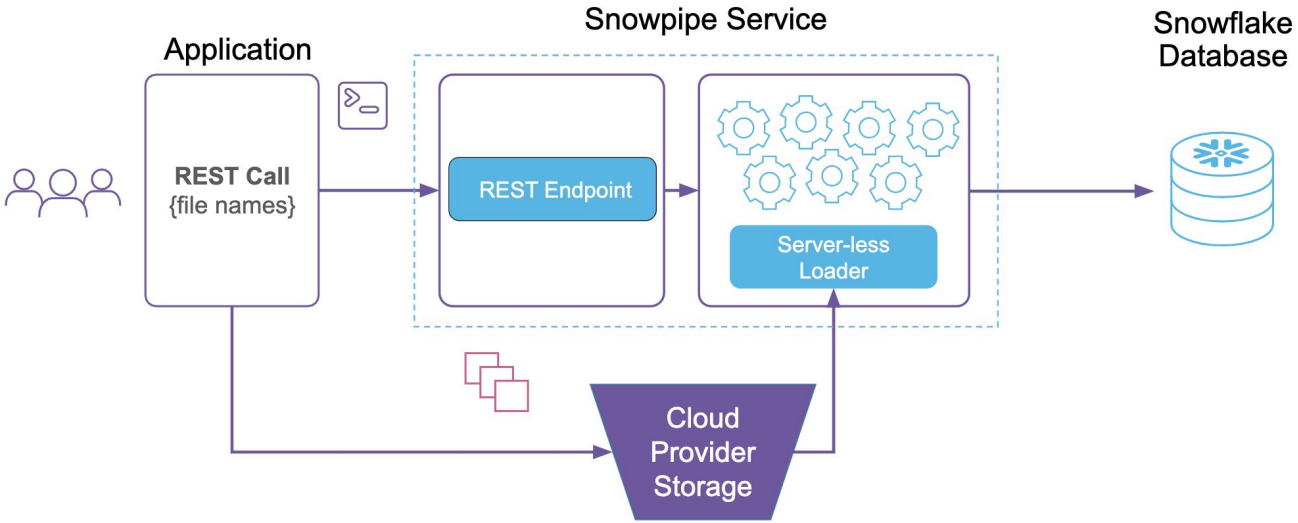
Snowpipe enables loading data from files as soon as they're available in a stage. This means you can load data from files in micro-batches, making it available to users within minutes, rather than manually executing COPY statements on a schedule to load larger batches.

- Named object contains a COPY statement used by Snowpipe
- Source stage for data files
    - Target table
- Loads data into tables continuously from an ingestion queue
- Can be paused/resumed, return status
- Best Practice: Size files between 10MB and 100MB (compressed) when staging files for ingest with Snowpipe

1. **Data files are loaded in a stage.**
2. **An S3 event notification informs Snowpipe via an SQS queue that files are ready to load. Snowpipe copies the files into a queue.**
3. A Snowflake-provided virtual warehouse loads data from the queued files into the target table based on parameters defined in the specified pipe.

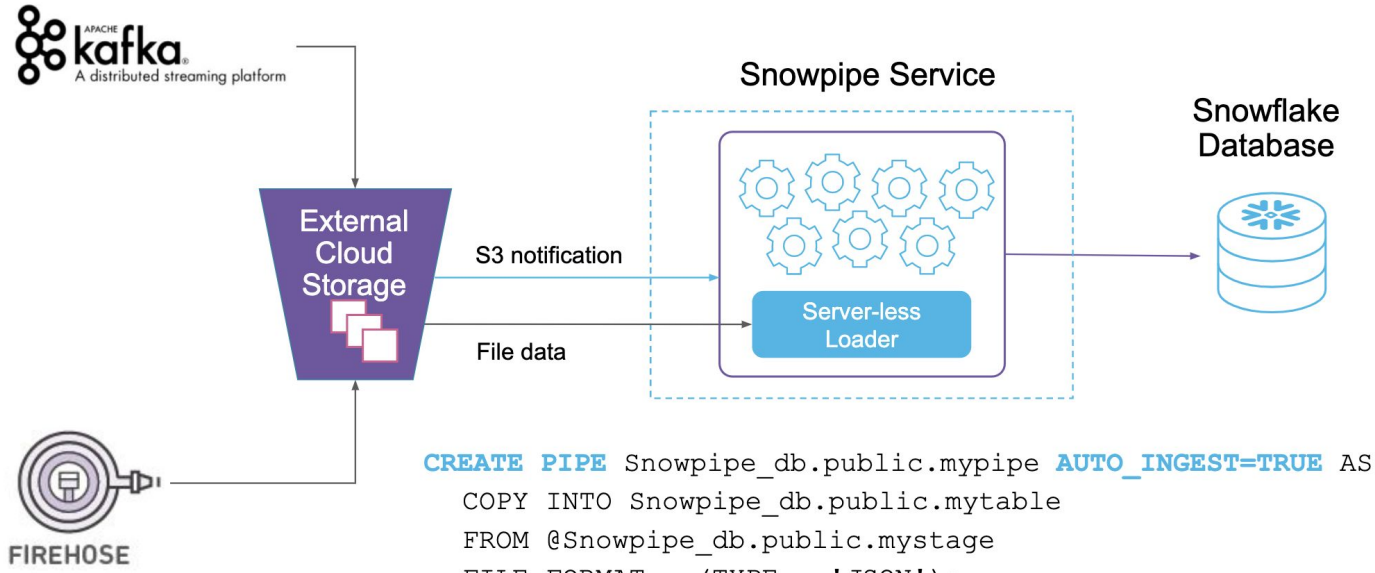


Snowpipe REST API



Auto ingest

- Snowflake gives an ARN that needs to be configured into the cloud platform's message queuing
- Currently only for pipes built on external stages




```
CREATE PIPE Snowpipe_db.public.mypipe AUTO_INGEST=TRUE AS
   COPY INTO Snowpipe_db.public.mytable
   FROM @Snowpipe_db.public.mystage
   FILE_FORMAT = (TYPE = 'JSON');
```

Unload to local file systems

- e.g. logs or data needed by non-Snowflake users
- load into the table stage; then unload that - COPY INTO @%region FROM region;
- Use COPY INTO
- Recommend using external named stage
- Can also unload directly by specifying the URI and any necessary credentials
- Data unloading consideration
    - File formats●Empty strings vs NULL values●Unloading relational table to semi-structured format●File splitting●Compress
- File formats
    - Any flat, delimited plain text format (CSV, TSC, etc.)
    - JSON
        - Data must be unloaded from a column of VARIANT data type
    - Parquet
        - Use a SELECT statement to unload a table to Parquet as multiple columns

Unload management

- Empty versus null
  - Enclose strings in double or single quotes:FIELD_OPTIONALLY_ENCLOSED_BY = 'character' | NONE
  - Determine how empty fields are handled:EMPTY_FIELD_AS_NULL = TRUE | FALSE
  - Convert SQL NULL values:NULL_IF = ('string1' [, 'string2' ...])
- File splitting
  - Using the option SINGLE, control whether the unload process will create a single file, or multiple files●Set MAX_FILE_SIZE to handle files larger than the default 16MB○The file size limit for single-file mode is 5GBCOPY INTO@mystage/myfile.csv.gz FROM mytableFILE_FORMAT = (type=csv compression='gzip') SINGLE=TRUE MAX_FILE_SIZE=4900000000
- Remove
  - Removes files that have been stored in an internal stage
  - REMOVE @%monthly_sales_agg PATTERN='.*data*.';

Functions

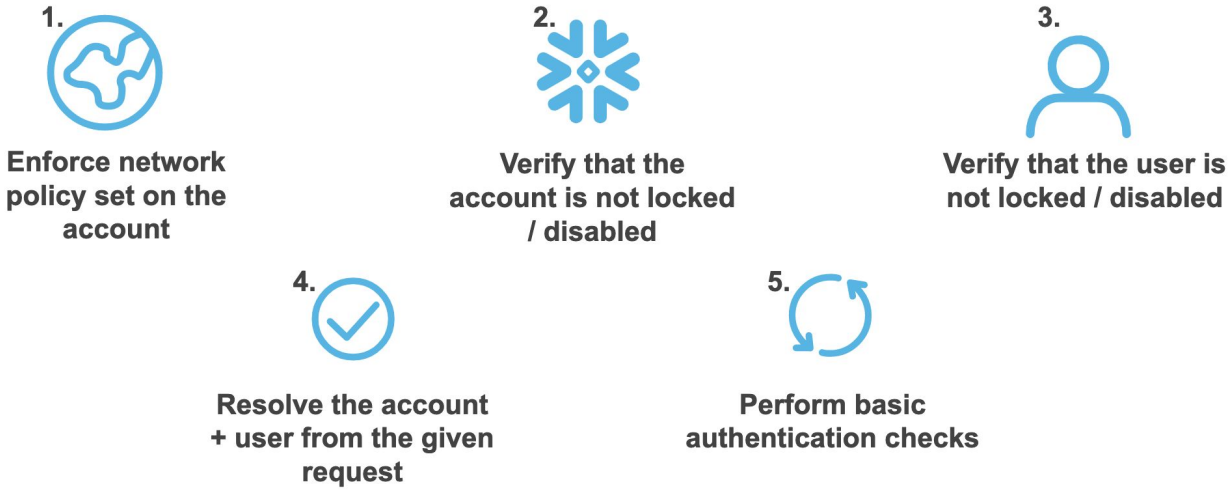
- Some aggregate functions don’t work over window, incl lead and lag?
- System functions - eg ABORT_SESSION, CANCEL_QUERY - can use programmatically
- User-defined functions
    - SQL or Javascript
    - Do not support DDL or DML
    - Stored procs can do most things though
    - Can secure so others can’t see source code
    - Return a single scalar value, (table result) a set of rows
    - Start and end code with $$ for SQL or ‘ for javascript
- Approximation - can use HyperLogLog, fast, within a few %
    - eg APPROX_COUNT_DISTINCT - counts of distinct cardinalities
    - MINHASH - Estimates the approximate similarity between two or more data sets
- Stored proc
    - javascript (case sensitive arguments) or SQL (not case sensitive)
    - CALL to invoke; within javascript snowflake.execute
- RANDOM returns the same value over multiple calls if seeded with a parameter:
    - SELECT RANDOM() AS random_variable;
    - SELECT RANDOM(100) AS random_fixed;

Security

- Access
  - communication secured & encrypted
  - TLS 1.2 encryption for all client communications
  - IP whitelisting
  - Support for AWS PrivateLink
  - rudimentary network mgmt - only 1 active policy
- Authentication
  - Password policy enforcement
  - Multi-factor authentication can be enabled on a per-user basis - recommended for admin accounts
  - Federated authentication (SAML 2.0) - integration with AD less effort since recent release
  - Support for Key-Pair authentication
- Authorisation
  - Flexible and granular authorisation controls
  - Role-based access control for granular permissions
  - Role-based access control for data and actions
  - Secure views and UDFs to protect information access
- Protection
  - Encrypted at rest an in transit (key rotation monthly); tri-secret secure (BYOK); time travel; failsafe
  - Micro-partitions go into failsafe when key updated => storage cost

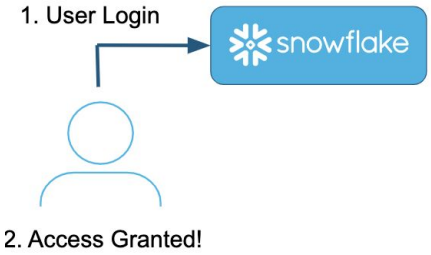
Security (cont)

- Infrastructure - as per cloud platform
  - Can now replicate remotely, eg from AWS Sydney to Azure Singapore
  - Soon will be able to failover the whole environment
- Encryption - Snowflake seamlessly encrypts on local machine before uploading
- S3 etc -

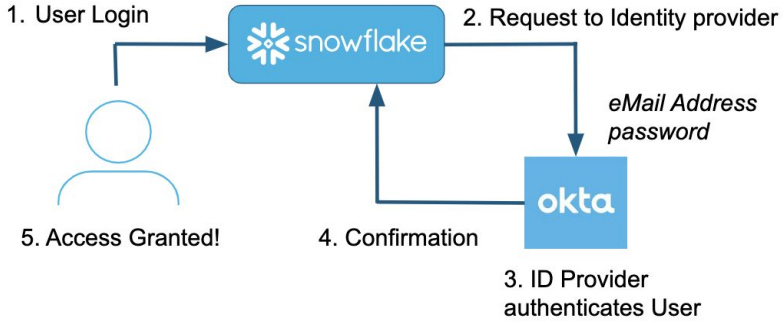


Authentication models

Snowflake username / password



Federated authentication



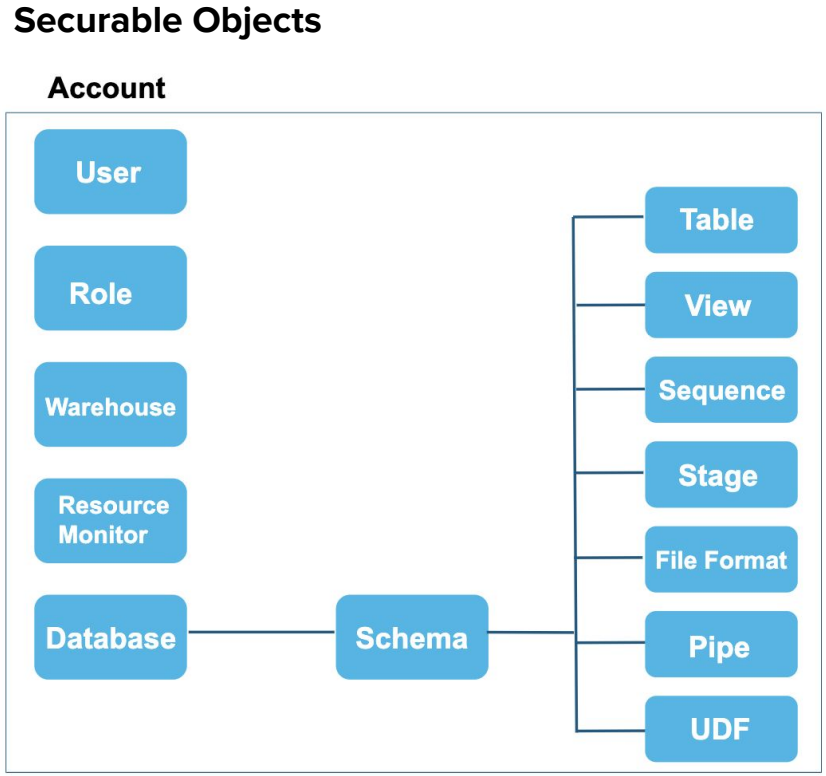
- OAuthAllows for securing authorizations on behalf of a user from a third party service / application
- KeyPair with JSON Web Tokens (JWT) signed using a public/private key pair with RSA encryption
- Minimum password policy: At least 8 characters; At least one digit; At least one upper/lower case

Network security

- Network Policy support for IP whitelisting and blacklisting
- Online certificate status protocol (OCSP) x.509 validation native to all Snowflake connectors
- Support for cloud provider connectivity options: like:
    - PrivateLink (AWS) - no traffic over the public internet
    - DirectConnect (AWS) - dedicated network connection on-premise to AWS
    - Azure options on roadmap
- Add a network policy to list allowed and blocked IPs
    - CREATE NETWORK POLICY; ALTER ACCOUNT ...
- Whitelist these ports:
    - 443 - general Snowflake traffic
    - 80 - for OCSP cache server, which listens for client comms
- SnowCD - utility to check setup

Access control and user management

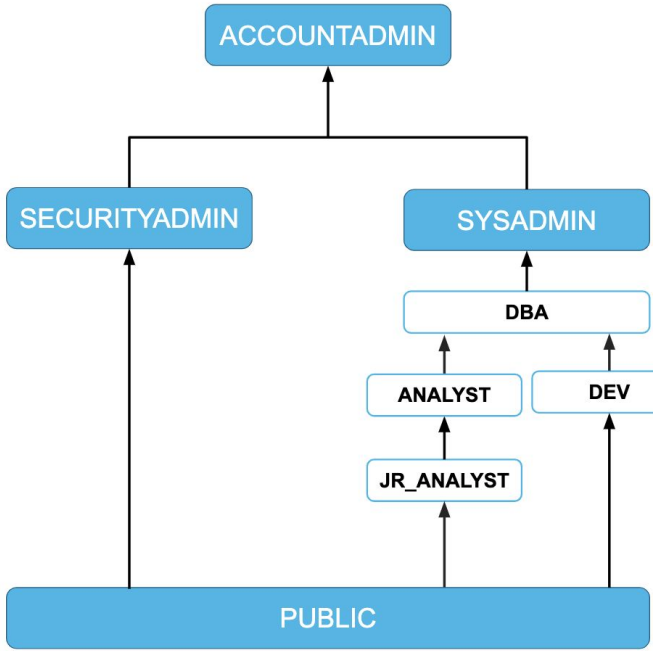
- Part of authorisation model
- Access control defines
  - Who can use which role and object
  - What operations can be performed
  - Who can alter access controls
- Roles - USE ROLE myrole; GRANT ROLE myrole to USER ANNA
  - Control server permissions, not just DB objects
  - GRANT USAGE OF WAREHOUSE xxx to ROLE prodRole (GRANT OPERATE allows changing the WH)
  - Stored procs can provide either creating or running role at run-time
- New views - grantstoroles; grantstousers
- GRANT SELECT, INSERT, DELETE ON ALL TABLES IN SCHEMA my_schema TO ROLE analyst, dba
- Resource monitor is used to notify and control events
- Need usage on DB and Schema for table privs to work



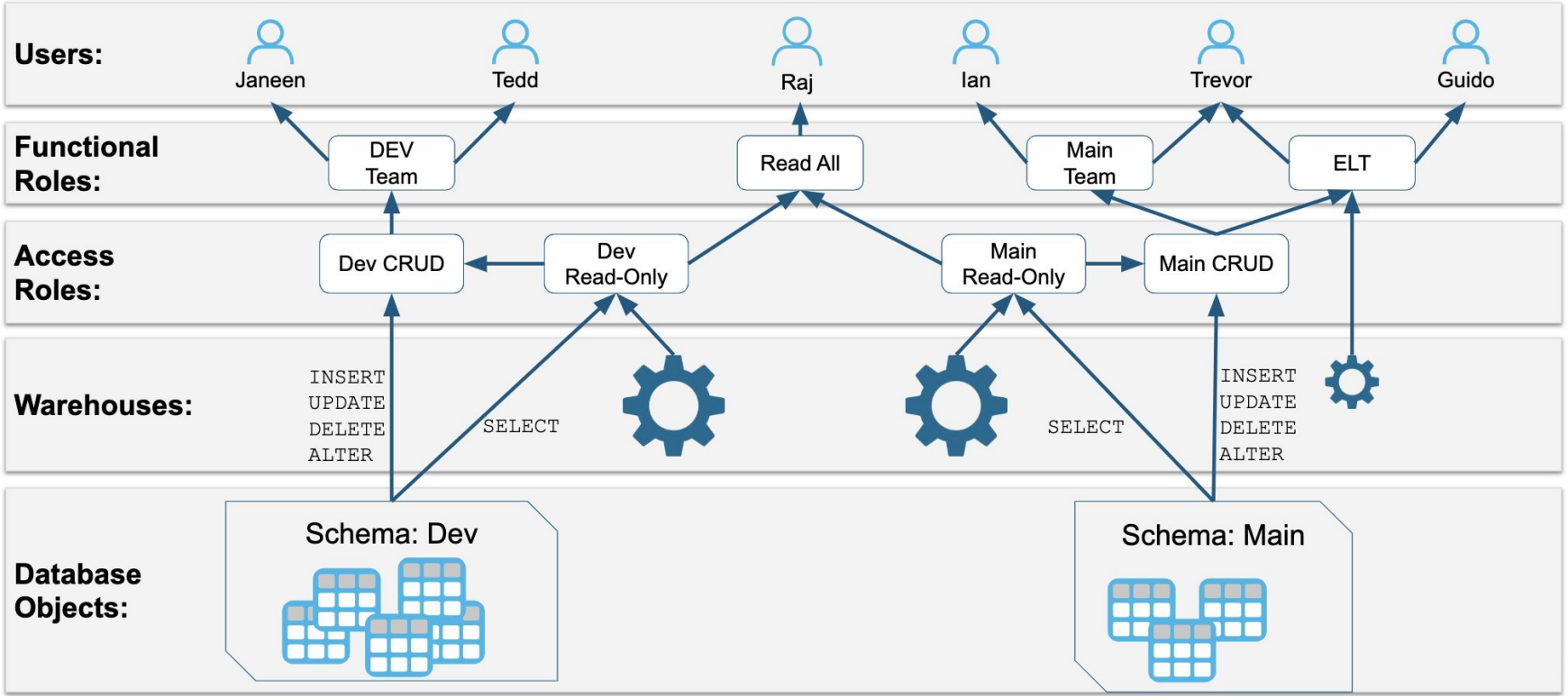
Roles (cont)

- Grant roles to roles
- System roles
  - ACCOUNTADMIN - has all privs - grant to limited users - never a user's default role!; don't run scripts
  - SECURITYADMIN - best to use only for managing users and roles
  - SYSADMIN - DBA
  - PUBLIC - minimal privs
- Recommend granting all roles to SYSADMIN as they're created
- GRANT OWNERSHIP ON DATABASE prd_db to ROLE dba;
- Managed Access Schema
  - Allows other users/roles to do DDL etc but not to grant access or ownership
- GRANT SELECT ON FUTURE TABLES IN SCHEMA - so role gets objects created in future immediately

ROLE HIERARCHY



Example role organisation



DEMO