Architecture

Snowflake Core Concepts and Keywords

KNOWLEDGE CHEATSHEET

SHARED DATA, INDEPENDENT COMPUTE

- **Storage** — Micro-partitions
- **Compute** — Warehouses
- **Cloud services** — Coordination

01 Snowflake Data Cloud features and architecture 25%

Three layers, shared data

- **Storage** holds managed table data in compressed, encrypted, columnar micro-partitions. Storage capacity grows independently of warehouse compute.
- **Compute** uses virtual warehouses to execute SQL, bulk loads, unloads and DML. Separate warehouses access the same data while isolating their compute workloads.
- **Cloud services** handles authentication, authorization, metadata, query parsing, optimization and coordination. A metadata operation can avoid warehouse execution while still using cloud services.
- **Multi-cluster shared data** combines centralized storage with independent massively parallel compute clusters. A warehouse is a compute resource, not a container for databases.
- Snowflake manages its service infrastructure, patching and statistics on **AWS, Azure and Google Cloud**. Each account belongs to a cloud region; an organization can contain accounts across clouds and regions.

Objects and table lifetimes

- The object hierarchy is **account > database > schema > object**. Tables, views, stages, file formats, streams, tasks and functions are schema objects; warehouses and account roles have account scope.
- Use **database.schema.object** for a fully qualified name. The database and schema form a namespace; session context resolves unqualified names.
- **Permanent tables** suit durable data. **Transient tables** persist across sessions but omit Fail-safe, making them useful for reproducible ETL data.
- **Temporary tables** belong to a session and disappear when it ends. A temporary table can hide a permanent table with the same name in that session; use deliberate naming and cleanup.
- On standard tables, **NOT NULL** is enforced; primary, unique and foreign keys generally describe relationships without enforcing them. **Hybrid tables** have different constraint enforcement rules.

Micro-partitions and metadata

- **Micro-partitioning** occurs automatically as data is loaded. Users do not define physical partitions or build conventional indexes on standard Snowflake tables.
- Micro-partitions represent roughly **50-500 MB of uncompressed data**, stored compressed. They are immutable: changes produce new partitions rather than editing existing files in place.

Architecture / Access & Security

Snowflake Core Concepts and Keywords

KNOWLEDGE CHEATSHEET

SHARED DATA, INDEPENDENT COMPUTE

- **Storage** — Micro-partitions
- **Compute** — Warehouses
- **Cloud services** — Coordination

- Metadata records column **minimums, maximums and distinct-value information**. The optimizer uses it to skip irrelevant partitions and read only needed columns.
- **Pruning** reduces scanned data. Overlapping value ranges can weaken pruning; natural load order and optional clustering keys affect how well filters isolate partitions.
- Metadata can satisfy certain simple counts or extrema without scanning table data. Filters, policies and query shape determine eligibility; do not assume every aggregate avoids a warehouse.

Editions and client tools

- **Standard** supplies core SQL, encryption, secure sharing and one-day Time Travel. **Enterprise** adds capabilities such as multi-cluster warehouses, extended retention, materialized views and advanced data governance.
- **Business Critical** adds stronger security and continuity options, including private connectivity and customer-managed key integration. **Virtual Private Snowflake** provides a dedicated service environment for stricter isolation requirements.
- **Snowsight** supports SQL and Python worksheets, charts, dashboards, object browsing and query monitoring. Sharing a worksheet or dashboard does not grant access to its underlying data.
- **SnowSQL** is a command-line SQL client with batch scripts, connection profiles and local file transfer. JDBC, ODBC and native connectors let compatible BI and application tools connect.
- **Partner Connect** provisions supported partner integrations. Distinguish data integration tools, BI tools, ML tools and governance/catalog tools by their purpose.
- **SnowCD** diagnoses network connectivity. **CURRENT_CLIENT()** identifies the current client; **CURRENT_VERSION()** reports the Snowflake service version.

02 Account access and security 20%

Roles, privileges and ownership

- **RBAC** assigns privileges to roles and roles to users. **Discretionary access control** gives each securable object an owner that normally controls grants on it.
- The **active primary role** authorizes object creation and owns new objects. A user default role is only a session default, not ownership of everything the user creates.
- Granting **child_role TO ROLE parent_role** makes the parent inherit the child's privileges. Owning a role and inheriting its privileges are separate relationships.
- Use **USE ROLE** to change the primary role and **USE SECONDARY ROLES ALL / NONE** to control additional active roles. Creation privileges come from the primary role.

Access & Security

Snowflake Core Concepts and Keywords

KNOWLEDGE CHEATSHEET

CONTROL ACCESS IN LAYERS

Authenticate Identity → **Authorize** Roles → **Protect** Policies → **Audit** History

- **OWNERSHIP** transfers to another role through GRANT OWNERSHIP. It is a single-owner privilege; dropping a custom role transfers its objects to the role performing the drop.
- Build custom job roles with **least privilege** and grant the top of the custom hierarchy to SYSADMIN. Keep ordinary object creation and automation away from ACCOUNTADMIN.

Administrative roles and grants

- **USERADMIN** creates users and roles. **SECURITYADMIN** inherits USERADMIN and has MANAGE GRANTS. **SYSADMIN** manages warehouses, databases and the object hierarchy granted to it.
- **ACCOUNTADMIN** combines account administration with SYSADMIN and SECURITYADMIN privileges. Restrict it to a small, resilient administrator group and protect interactive access with MFA.
- **ORGADMIN** handles organization-level account operations. **PUBLIC** is available to every user; grant to it only what all users should receive.
- Read access usually needs **USAGE on the database and schema**, **SELECT on the table**, and **USAGE on the warehouse** when execution needs compute. Parent USAGE does not itself grant table access.
- **ALL TABLES** grants affect existing tables; **FUTURE TABLES** grants affect subsequently created tables. Use both when a role needs current and future access.
- Use **SHOW GRANTS TO ROLE** for privileges a role receives, **SHOW GRANTS OF ROLE** for its users and parent roles, and **SHOW GRANTS ON** for an object's grants. **REVOKE ROLE** removes a role assignment.

Centralized grants and operations

- In a **managed access schema**, the schema owner or a role with global **MANAGE GRANTS** controls grants on contained objects. Individual object owners retain ownership but lose independent grant decisions.
- Warehouse **USAGE** permits query use; **OPERATE** permits operational actions such as resume, suspend and abort; **MODIFY** permits property changes such as resizing; **MONITOR** exposes activity and statistics.
- A **database role** groups privileges within one database and is granted to account roles for use. Providers can grant database roles to shares to organize consumer access.
- Separate **policy administration**, object ownership and data access. Owning a table does not automatically grant unrestricted authority to change governance policies.

Access & Security

Snowflake Core Concepts and Keywords

KNOWLEDGE CHEATSHEET

CONTROL ACCESS IN LAYERS

Authenticate Identity → **Authorize** Roles → **Protect** Policies → **Audit** History

Identity and connection controls

- **Authentication** proves identity; **authorization** decides permitted actions. **SAML SSO** delegates sign-in to an identity provider through a security integration; Snowflake roles still control data access.
- **MFA** adds an independent authentication factor. Use authentication policies to govern permitted methods and client access; supported drivers can cache MFA tokens when allowed.
- **Key-pair authentication** keeps the private key with the client and registers the public key on the user, for example with **RSA_PUBLIC_KEY**. It is useful for supported automated clients.
- **OAuth** authorizes clients using tokens. **SCIM** provisions users and groups; it is not a user sign-in protocol. REST API authentication depends on the API and supported token method.
- **Network policies** restrict connection sources using allowed and blocked addresses or network rules. A blocked address wins over the same allowed address; a user policy overrides the account policy.
- **PrivateLink / Private Service Connect** supplies supported private network paths. **Session policies** control idle session behavior; neither control replaces role permissions.

Encryption and sensitive data

- Snowflake encrypts managed data **at rest** and uses **TLS in transit**. Hierarchical encryption keys separate account, table and file protection; rotation changes active keys, while rekeying re-encrypts older data.
- **Tri-Secret Secure** combines a Snowflake-managed key with a customer-managed key. It differs from key-pair authentication, which authenticates a client.
- **Dynamic masking** changes column values at query time according to policy conditions. **Conditional masking** can use other columns; **row access policies** decide which rows a query may see.
- **External tokenization** substitutes tokens for sensitive values before ingestion using a supported service. Encryption and masking solve different exposure problems.
- **Classification** identifies sensitive data. **Tags** attach governed metadata for discovery, compliance and cost allocation; tag-based masking connects labels with protection policies.
- **POLICY_REFERENCES** identifies policy attachments; **MASKING_POLICIES** inventories masking policies. **GET_DDL** retrieves object definitions subject to the caller's privileges.

Access & Security / Performance

Snowflake Core Concepts and Keywords

KNOWLEDGE CHEATSHEET

CONTROL ACCESS IN LAYERS

Authenticate Identity → **Authorize** Roles → **Protect** Policies → **Audit** History

Usage history and auditing

- Each database has an **INFORMATION_SCHEMA** with metadata views and table functions. **SNOWFLAKE.ACCOUNT_USAGE** provides account-wide history, often including dropped objects, with view-specific latency and retention.
- **QUERY_HISTORY** records execution, duration, warehouse and errors. **LOGIN_HISTORY** records authentication events and client addresses. **ACCESS_HISTORY** supports object/column access auditing and write lineage.
- **WAREHOUSE_LOAD_HISTORY** shows running and queued load. **WAREHOUSE_METERING_HISTORY** tracks warehouse credits; **METERING_HISTORY** covers service usage. Usage is not always the same as billed credits.
- **TABLE_STORAGE_METRICS** separates active, Time Travel, Fail-safe and clone-retained storage. **SHOW TABLES** gives a simpler current view; it does not expose every storage category.
- Account Usage query and login histories support longer investigations than short-window Information Schema functions. **ACCESS_HISTORY** retains a year of history and requires Enterprise or higher.
- Grant suitable **SNOWFLAKE database roles**, such as GOVERNANCE_VIEWER for policy information, rather than broad administrative access. MONITOR USAGE supports delegated usage visibility.

03 Performance concepts 15%

Warehouse size and lifecycle

- **Scale up** increases resources per cluster for complex SQL and memory pressure. **Scale out** adds clusters for concurrency. A single query is not distributed across every cluster in a multi-cluster warehouse.
- Resize a warehouse while running or suspended. Running queries continue with their allocated resources; a suspended warehouse adopts the new size when resumed. **WAIT_FOR_COMPLETION** waits for resizing to finish.
- **AUTO_SUSPEND** stops an idle warehouse after the configured interval. **AUTO_RESUME** restarts it when needed. **INITIALLY_SUSPENDED** controls its initial state.
- A value of **0 or NULL for AUTO_SUSPEND** disables automatic suspension. Choose idle timeouts by balancing idle credits, restart latency and local cache reuse.
- Benchmark representative queries on several sizes. Separate loading and analytics warehouses when their concurrent work competes for resources; size by workload rather than total account storage.
- **Snowpark-optimized warehouses** provide more memory for workloads such as ML training. Increasing cluster count addresses concurrency rather than the memory available to one query.

Performance

Snowflake Core Concepts and Keywords

KNOWLEDGE CHEATSHEET

MATCH THE REMEDY TO THE SYMPTOM

- **Queuing** — Scale out
- **Memory pressure** — Scale up
- **Excess scanning** — Improve pruning
- **Repeated work** — Reuse results

Concurrency and scaling policies

- A **multi-cluster warehouse** uses equally sized clusters. **MIN_CLUSTER_COUNT < MAX_CLUSTER_COUNT** enables auto-scaling; equal counts above one select maximized mode with that many clusters.
- **Standard** scaling prioritizes responsiveness and starts extra clusters more readily. **Economy** favors fuller utilization and credit conservation, accepting more queuing before adding capacity.
- Raising **MAX_CLUSTER_COUNT** permits more scale-out but does not force all clusters to run. Raising **MIN_CLUSTER_COUNT** keeps more capacity ready at additional cost.
- When capacity is full, new queries **queue**. Use warehouse load history to distinguish sustained concurrency from occasional heavy queries; scale out or isolate workloads when queuing dominates.
- **MAX_CONCURRENCY_LEVEL** controls per-cluster query concurrency. **STATEMENT_QUEUED_TIMEOUT_IN_SECONDS** limits queue wait; **STATEMENT_TIMEOUT_IN_SECONDS** limits statement duration. A timeout changes tolerance, not capacity.

Three different caches

- **Persisted result cache** stores query output separately from a warehouse. Eligible repeated SQL can reuse results without warehouse execution when data, privileges and relevant settings still match.
- Results normally persist for **24 hours**. Eligible reuse renews the retention window, up to **31 days from initial execution**; changed data, changed SQL or nondeterministic expressions can prevent reuse.
- **Warehouse data cache** holds previously accessed table data on local SSD. It speeds scans, requires running compute, and is lost on suspension; downsizing can discard part of the cache.
- **Metadata** supports planning, pruning and eligible metadata-only answers. A high **percentage scanned from cache** in Query Profile refers to warehouse data cache, not persisted result reuse.
- **RESULT_SCAN** treats a previous command's result as a table for further SQL. Permission to see query history does not automatically permit reading another user's result set.

Performance

Snowflake Core Concepts and Keywords

KNOWLEDGE CHEATSHEET

MATCH THE REMEDY TO THE SYMPTOM

- **Queuing** — Scale out
- **Memory pressure** — Scale up
- **Excess scanning** — Improve pruning
- **Repeated work** — Reuse results

Read the Query Profile

- **EXPLAIN** shows an estimated plan without executing the query. **Query Profile** shows actual operators, row flows, timing and statistics for an executed query.
- Start with the **most time-consuming operators**. Compare input and output rows, bytes scanned, partitions scanned versus total, and time spent processing, synchronizing or doing I/O.
- For a selective filter, scanning nearly every partition can indicate weak **pruning**. A full scan can still be appropriate for a query that genuinely needs the whole table.
- **Spilling** occurs when intermediate data exceeds memory: local storage first, then remote storage if needed. Remote disk I/O can expose expensive spill activity; inspect spill-byte statistics before choosing a remedy.
- Reduce spill with a larger warehouse, fewer unnecessary columns, earlier selective processing or smaller batches. More clusters do not enlarge a single query's memory allocation.
- An **exploding join** produces far more rows than expected, often from missing or nonunique join conditions. **UNION** adds duplicate elimination; **UNION ALL** avoids that work when duplicates are acceptable.

Choose the right optimization

- A **clustering key** helps organize a large table around frequent selective filters or joins. Choose useful cardinality; expressions such as a timestamp-to-date conversion can reduce excessive distinctness.
- **SYSTEM$CLUSTERING_INFORMATION** and **SYSTEM$CLUSTERING_DEPTH** describe overlap and depth. Lower overlap generally improves pruning; a key on every table is unnecessary.
- **Automatic Clustering** maintains the organization using Snowflake-managed compute and can add storage churn. Use **ALTER TABLE ... DROP CLUSTERING KEY** when the key no longer benefits the workload.
- **Search Optimization Service** maintains persistent search access paths for supported highly selective predicates, including point lookups. It adds maintenance and storage costs rather than accelerating every query.
- **Materialized views** persist selected query results and are maintained automatically. They help repeated expensive work over relatively stable data but add storage and refresh cost; their definitions have restrictions, including a single base table and no joins.
- **Query Acceleration Service** offloads eligible portions of scan-heavy queries to shared serverless resources. It can help atypically expensive analyses; it does not replace multi-cluster concurrency management.

Performance / Loading & Unloading

Snowflake Core Concepts and Keywords

KNOWLEDGE CHEATSHEET

MATCH THE REMEDY TO THE SYMPTOM

- **Queuing** — Scale out
- **Memory pressure** — Scale up
- **Excess scanning** — Improve pruning
- **Repeated work** — Reuse results

Credit consumption and limits

- Warehouse credits depend on **size, running clusters and running time**, including idle time before suspension. Billing is per second with a **60-second minimum** on each start or resume; new scale-up resources also have a minimum.
- Each successive size generally doubles the hourly credit rate. Calculate each cluster's running intervals separately; a suspended warehouse consumes no warehouse credits.
- **Serverless features** have separate charges. For cloud services, the daily adjustment offsets up to **10% of warehouse credits**; only excess cloud-services usage is billed. Serverless usage does not enlarge that adjustment.
- **Resource monitors** set credit quotas and threshold actions for assigned warehouses or the account. **NOTIFY** alerts; **SUSPEND** lets active statements finish; **SUSPEND_IMMEDIATE** cancels them.
- One warehouse has at most one directly assigned monitor; an account monitor can also constrain it. Monitors do not cap all serverless spending, and threshold detection can allow some overshoot.
- **Storage charges** include compressed managed data, internal stage files and retained historical data. External-stage storage is billed by its cloud provider; cross-region or cross-cloud transfers can add transfer costs.

04 Data loading and unloading 10%

Stage locations and commands

- An **internal stage** stores files in Snowflake-managed storage. Reference a user stage as **@~**, a table stage as **@%table_name**, or a named stage as **@stage_name**.
- User and table stages exist implicitly. A **named stage** is a reusable schema object. An **external stage** references cloud object storage such as S3, Azure or Google Cloud Storage.
- **PUT** uploads local files to an internal stage through supported clients. It normally compresses eligible files with GZIP; the default fully encrypted internal-stage path encrypts locally before transfer over TLS.
- **LIST / LS** lists stage files. **GET** downloads files from an internal stage to the client. For external storage, use the cloud tooling or appropriate file-access mechanism.
- **REMOVE** deletes staged files; **PURGE = TRUE** removes successfully loaded source files during COPY. Dropping an external stage leaves external files intact; dropping an internal stage deletes its files without Time Travel recovery.
- Use a **storage integration** and least-privilege cloud permissions for external stages. Loading needs cloud read access; unloading needs write access as well as appropriate bucket/container access.

Loading & Unloading

Snowflake Core Concepts and Keywords

KNOWLEDGE CHEATSHEET

FILE MOVEMENT AND LOADING

Local files PUT → **Stage** COPY INTO table → **Table** COPY INTO stage → **Internal stage** GET

File formats and load design

- Supported load formats include **CSV, JSON, Avro, ORC, Parquet and XML**. CSV includes configurable delimiters, so it also covers TSV-style data. Parquet is a columnar format.
- A **named file format** centralizes reusable parsing settings; inline options also work. Explicit COPY settings override inherited stage settings. CSV is the default type when no alternative is defined.
- CSV options include **FIELD_DELIMITER**, **SKIP_HEADER**, **FIELD_OPTIONALLY_ENCLOSED_BY**, **NULL_IF** and **EMPTY_FIELD_AS_NULL**. Quote delimiter-containing fields and distinguish empty strings from nulls deliberately.
- Use **UTF-8** as the normal interchange encoding. Match the parser to the actual file structure and compression rather than relying only on filename extensions.
- For bulk loads, target roughly **100-250 MB compressed per file** as a starting point and provide enough files for parallelism. Avoid one enormous file or huge numbers of tiny files.
- Organize staged data into meaningful **paths/prefixes** and load only the relevant path. Broad regular-expression PATTERN matching can add listing overhead. Benchmark warehouse size against file count and format.

Bulk COPY and validation

- **COPY INTO table FROM @stage/path** loads staged records using a running warehouse. It appends rows; a file containing changed business keys does not automatically update earlier rows.
- COPY load transformations can **reorder or omit columns and cast values** through a SELECT. They are limited transformations; use subsequent SQL for joins, aggregations and richer filtering.
- **VALIDATION_MODE** checks files without loading them. Modes include **RETURN_n_ROWS**, **RETURN_ERRORS** and **RETURN_ALL_ERRORS**; validation has restrictions when COPY includes transformations.
- **ON_ERROR** determines load-error handling, such as aborting or continuing. Inspect COPY results and **COPY_HISTORY** for partial loads and failures; **VALIDATE(table, JOB_ID => ...)** returns errors from a previous load.
- Table-level load metadata helps skip previously loaded files. **FORCE = TRUE** bypasses that protection and can duplicate data; **LOAD_UNCERTAIN_FILES** deals with files whose prior status cannot be established.
- Load-history deduplication concerns **files**, not duplicate business rows. Use staging tables and **MERGE** for business-key inserts and updates; clean up staged files only after checking success.

Loading & Unloading

Snowflake Core Concepts and Keywords

KNOWLEDGE CHEATSHEET

FILE MOVEMENT AND LOADING

- **Local files** — PUT → **Stage** — COPY INTO table → **Table** — COPY INTO stage → **Internal stage** — GET

Continuous ingestion

- **Snowpipe** uses Snowflake-managed compute to load arriving staged files in small batches. A **pipe** stores the COPY definition; it does not need a dedicated user warehouse.
- **Auto-ingest** responds to cloud file notifications. The **Snowpipe REST API**, including insertFiles, can submit staged file paths; notifications identify files rather than carrying table rows.
- A pipe tracks its load history and can be paused or resumed. Recreating it resets pipe metadata, so coordinate recovery carefully to avoid missed files or reloading.
- Balance **ingestion latency, file size and per-file overhead**. Bulk COPY suits large existing batches; Snowpipe suits continuous arrivals; **Snowpipe Streaming** ingests row data without staging files.
- The Kafka connector's classic raw record layout uses **RECORD_CONTENT** and **RECORD_METADATA**, both VARIANT columns. Connector configuration and ingestion mode determine the actual target mapping.

Unload and retrieve files

- **COPY INTO @stage/path FROM table** or **FROM (SELECT ...)** unloads data using warehouse compute. Targets can be internal stages, external stages or supported cloud locations.
- Unload formats are **CSV, JSON and Parquet**. For JSON, use **OBJECT_CONSTRUCT(*)** in the SELECT to turn each relational row into one object column.
- **SINGLE = FALSE** is the default and permits parallel output files. **SINGLE = TRUE** requests one file; **MAX_FILE_SIZE** guides file sizing without making parallel output files exact-sized.
- **PARTITION BY** routes output into paths based on expression values. **INCLUDE_QUERY_ID** makes unload filenames distinct across runs; **OVERWRITE** affects matching output names rather than cleaning every old file.
- Configure compression and null/empty-string representation for the target format. Parquet avoids the text-format conversion issues associated with floating-point CSV/JSON output.
- Use **VALIDATION_MODE = RETURN_ROWS** to return unload-query rows instead of writing files. After unloading to an internal stage, use **LIST** to inspect files and **GET** to download them.

Transformations

Snowflake Core Concepts and Keywords

KNOWLEDGE CHEATSHEET

TRANSFORM AND AUTOMATE

Parse VARIANT → **Expand** FLATTEN → **Transform** SQL / Snowpark → **Automate** Streams / tasks

05 Data transformations 20%

VARIANT and JSON paths

- **VARIANT** preserves a value and its type, including nested **OBJECT** key/value pairs and **ARRAY** elements. **PARSE_JSON** parses JSON text; converting a string with TO_VARIANT does not parse its JSON structure.
- Traverse with **src:customer.name** or bracket notation. Arrays are zero-indexed: **src:elements[0].name** reads the first element. Double-quote JSON keys that contain spaces or special characters.
- JSON **element names are case-sensitive**. SQL column names follow SQL identifier rules: unquoted identifiers are case-insensitive; quoted identifiers preserve case.
- Cast extracted values, for example **src:price::NUMBER**, when typed SQL operations are needed. **TYPEOF** reports the stored type; predicates such as **IS_ARRAY** test it.
- **SQL NULL** and **JSON null** differ: PARSE_JSON(NULL) returns SQL NULL, while PARSE_JSON('null') returns a VARIANT null. **IS_NULL_VALUE** detects JSON null; **STRIP_NULL_VALUE** converts it to SQL NULL.
- **STRIP_OUTER_ARRAY** loads an outer JSON array's elements as separate rows. **STRIP_NULL_VALUES** removes null-containing elements during load. Project frequently used dates and timestamps into native columns when beneficial.

Flatten, sample and summarize

- **LATERAL FLATTEN(INPUT => src:items)** expands nested elements while retaining columns from the preceding row. Output includes **SEQ, KEY, PATH, INDEX, VALUE and THIS**; RECURSIVE traverses nested containers.
- **SAMPLE / TABLESAMPLE (10)** is a percentage sample; **SAMPLE (10 ROWS)** requests a fixed row count, limited by table size. **BERNOULLI / ROW** samples rows; **SYSTEM / BLOCK** samples blocks.
- Sampling without a seed can vary between executions. Supported seeded block sampling improves repeatability; a LIMIT clause is not a random sample.
- **Window functions** use OVER with PARTITION BY and ORDER BY; **QUALIFY** filters their results. **INTERSECT** returns common rows, **MINUS / EXCEPT** returns differences, and UNION combines sets.
- **SPLIT_TO_TABLE** turns delimited text into rows through TABLE(...). **SPLIT_PART** returns a chosen substring and an empty string for an out-of-range part number.
- **HyperLogLog / APPROX_COUNT_DISTINCT** estimates cardinality efficiently. **JAROWINKLER_SIMILARITY** scores string similarity. Recursive **CTEs** and **CONNECT BY** traverse hierarchies.

Transformations

Snowflake Core Concepts and Keywords

KNOWLEDGE CHEATSHEET

TRANSFORM AND AUTOMATE

Parse VARIANT → **Expand** FLATTEN → **Transform** SQL / Snowpark → **Automate** Streams / tasks

Views, UDFs and procedures

- A **standard view** stores a query definition rather than a separate result dataset. A **secure view** hides its definition from unauthorized users and restricts optimizations that could reveal protected data.
- Use secure views for privacy and controlled sharing. They may trade some performance for protection; they do not create an independently encrypted copy of the underlying table.
- A **scalar UDF** returns one value; a **UDTF** returns rows and is called through TABLE(...). Function overloading uses the same name with different argument signatures.
- **Stored procedures** run procedural logic and database operations through CALL. They can use multiple SQL statements; **caller's rights** uses caller privileges, while **owner's rights** delegates the owner's permitted authority.
- UDFs and procedures support languages such as **SQL, JavaScript, Python, Java and Scala**, with language-specific capabilities. Dollar-quoted **$$** bodies avoid escaping embedded quotes.
- **Snowpark** expresses transformations in supported programming languages and pushes execution into Snowflake. **External functions** invoke remote services through integrations rather than running their service code inside Snowflake.

Change data and orchestration

- A **stream** exposes change data since its offset; it does not store a duplicate table. **METADATA$ACTION** contains INSERT or DELETE, and **METADATA$ISUPDATE** identifies update-related records.
- **Standard streams** track inserts, deletes and updates. **Append-only streams** focus on inserts; **insert-only streams** support external-table change tracking with different semantics.
- A plain **SELECT** does not consume a stream. Committed DML that uses it advances its offset. Consume before **STALE_AFTER**; retention-extension settings reduce staleness risk without retaining changes forever.
- A **task** runs SQL, a scripting block or a procedure call on a schedule or supported trigger. Task graphs express dependencies. Tasks can use a chosen warehouse or Snowflake-managed serverless compute.
- Task administration needs appropriate schema and execution privileges. Use **OPERATE** to suspend or resume a task, **TASK_HISTORY** for executions and **CURRENT_TASK_GRAPHS** for scheduled or running graph instances.
- **Alerts** periodically evaluate a condition and run an action. Snowflake Scripting supports blocks, variables and loops: WHILE tests before iteration, REPEAT tests after, and LOOP needs an exit condition such as BREAK.

Transformations

Snowflake Core Concepts and Keywords

KNOWLEDGE CHEATSHEET

TRANSFORM AND AUTOMATE

Parse VARIANT → **Expand** FLATTEN → **Transform** SQL / Snowpark → **Automate** Streams / tasks

Files and directory tables

- **Unstructured data** remains in staged files such as documents or media; SQL manages metadata and access. UDF or procedure handlers can process file contents; external functions can delegate processing to a service.
- A **directory table** is metadata layered on an internal or external named stage, not a separate securable table. Enable it with DIRECTORY = (ENABLE = TRUE); stage privileges govern access.
- Query **DIRECTORY(@stage)** for file metadata such as RELATIVE_PATH, SIZE, LAST_MODIFIED and FILE_URL. Refresh metadata manually or configure supported automatic refresh.
- **BUILD_STAGE_FILE_URL** returns a permanent file URL whose use requires authorization. **BUILD_SCOPED_FILE_URL** returns scoped access valid for the caller until its persisted-result window expires.
- **GET_PRESIGNED_URL** returns temporary bearer access: anyone possessing the URL can use it until expiration. Secure views can expose scoped or pre-signed access for sharing files.
- For file URLs, **USAGE on an external stage** or **READ on an internal stage** authorizes access. Use **SNOWFLAKE_SSE** for internal-stage URL access; **SNOWFLAKE_FULL** also applies client-side encryption.
- **External tables** query externally stored files through refreshed metadata and are read-only to DML. They differ from directory tables, which catalog files rather than expose their records as table data.

SQL and session essentials

- **USE WAREHOUSE**, **USE DATABASE**, **USE SCHEMA** and **USE ROLE** establish session context. Explicit client connection settings normally override user defaults for the new connection.
- **SET variable = value** creates a session variable referenced as **$variable**. **LAST_QUERY_ID()** identifies recent statements; a positive index counts from the start of the session and a negative index counts backward.
- **CREATE TABLE ... LIKE** copies a table definition without its rows. **CTAS** executes a query to create populated data; **MERGE** applies matched and unmatched business-key logic.
- **NUMBER / DECIMAL / NUMERIC** are fixed-point types. **FLOAT / DOUBLE / REAL** are approximate floating-point types. Use VARCHAR for character data and BINARY for binary values instead of unsupported CLOB/BLOB declarations.
- **GEOGRAPHY** models Earth coordinates; **GEOMETRY** models planar coordinates. Match spatial functions and coordinate systems to the intended analysis.
- SnowSQL **config** profiles store connection options; **-a** selects an account and **-c** a named connection. **output_file** and **output_format** control client output, while quiet/header settings help produce machine-readable exports.

Protection & Sharing

Snowflake Core Concepts and Keywords

KNOWLEDGE CHEATSHEET

CHOOSE THE PROTECTION MECHANISM

- **History** — Time Travel
- **Snapshot** — Clone
- **Collaboration** — Share
- **Continuity** — Replication

06 Data protection and data sharing 10%

Time Travel and Fail-safe

- **Time Travel** queries, clones or restores historical data within retention. Use **AT / BEFORE** with TIMESTAMP, OFFSET or STATEMENT; **UNDROP** restores supported dropped objects such as tables, schemas and databases.
- Default retention is **one day**. Standard permits 0-1 day; Enterprise or higher permits up to **90 days for permanent objects**. Temporary and transient tables remain limited to 0-1 day.
- **DATA_RETENTION_TIME_IN_DAYS** is inherited unless overridden. Account **MIN_DATA_RETENTION_TIME_IN_DAYS** sets a floor; the effective value uses the higher setting within supported limits.
- **Fail-safe** follows Time Travel for permanent table data and lasts **seven days**. It is a Snowflake-operated last-resort recovery mechanism, not a user-queryable backup or configurable extension.
- Temporary and transient tables have **no Fail-safe**. Temporary tables cannot be recovered after their session ends. Retained historical data consumes storage, and dropping a table does not immediately eliminate all its storage charges.

Zero-copy clones

- **CREATE ... CLONE** creates a point-in-time logical copy of supported tables, schemas or databases. Initially, source and clone reference the same existing micro-partitions without copying their table data.
- A clone is **independent**: later source changes do not synchronize into it. New or changed partitions consume storage; shared partitions remain needed while any object references them.
- Use clones for **development, testing and snapshots**. A historical clone must fall within Time Travel retention; cloning is not account-to-account replication.
- A table clone can use **COPY GRANTS** to copy privileges other than OWNERSHIP. Container cloning preserves supported child-object grants, while the new top-level object needs its own access design.
- Check object-specific cloning behavior. Fully qualified view or procedure references can still point at the source database; cloning a definition does not rewrite every embedded name or copy external files.

Protection & Sharing

Snowflake Core Concepts and Keywords

KNOWLEDGE CHEATSHEET

CHOOSE THE PROTECTION MECHANISM

- **History** — Time Travel
- **Snapshot** — Clone
- **Collaboration** — Share
- **Continuity** — Replication

Provider and consumer sharing

- A **share** grants selected database objects to consumers. Same-region secure sharing exposes live data through metadata without copying it into each consumer's storage.
- The **provider** stores and controls the source data; a normal **consumer** uses its own compute to query it. A shared database is read-only, but its data can participate in queries and joins with local data.
- Providers grant **USAGE on database/schema** and appropriate object privileges, such as **SELECT on tables and secure views**, to a share. Supported secure UDFs and database roles can also participate.
- Only explicitly shared objects are accessible. New objects require grants; revoking access removes consumer visibility. Consumers cannot modify provider objects or simply re-share the inbound share.
- A secure view referencing other databases needs **REFERENCE_USAGE** on those databases in addition to the sharing grants. **CURRENT_ACCOUNT()** can help tailor shared results to a consumer account.
- **Reader accounts** let recipients without their own Snowflake account query provider data. The provider creates and pays for these accounts' compute; they are restricted consumption environments.

Listings, replication and continuity

- **Direct shares** target known accounts. **Listings** package data access with descriptive and commercial information for private distribution or the **Snowflake Marketplace**.
- A **Data Exchange** governs private collaboration among invited member accounts. An account can provide some datasets and consume others; membership does not automatically grant every listing.
- **IMPORT SHARE** and **CREATE DATABASE** support importing shared data. Consumers receive access through database roles or imported privileges according to the provider's sharing model.
- Cross-region or cross-cloud delivery needs **replication** or supported **Cross-Cloud Auto-Fulfillment** for listings. Publishing listing metadata alone does not place the underlying data in every region.
- **Replication** refreshes a secondary copy and incurs destination storage and potential transfer costs. **Failover groups** add supported promotion for disaster recovery; they serve a different purpose from historical recovery.
- **Client Redirect** uses a connection object to redirect clients during continuity operations. Coordinate promotion, client endpoints, refresh lag and recovery objectives; ordinary zero-copy cloning does not solve regional outages.