Account & Security

SnowPro Advanced: Architect

KNOWLEDGE CHEATSHEET

SECURITY BOUNDARIES

- **Identity** — Authenticate
- **Roles** — Authorize
- **Policies** — Protect
- **Network** — Restrict

01 Account and security

Account strategy and object hierarchy

- **Organization > account > database > schema > object** describes the main containment hierarchy. Tables, views, stages and tasks are schema objects; warehouses, users, account roles and integrations belong to the account.
- **Single-account environments** simplify cloning, references and administration. Separate production, test and development databases, warehouses and roles; grant no production access to development roles unless required.
- **Multiple accounts** support separate regions, clouds, editions, identity configurations and stronger administrative isolation. Budget for replication, deployment automation and duplicated governance configuration.
- **Tenancy** can use an account per tenant, objects per tenant, or shared tables with tenant filters. Choose the boundary from legal isolation, customization, administrative scale and cost requirements.
- **Parameters** have account, user, session or object scopes depending on the parameter. More specific supported settings can override inherited defaults; inspect effective values rather than assuming every parameter follows one hierarchy.

Roles, ownership and grants

- **RBAC** assigns privileges to roles and roles to users or other roles. Grant an access role to a functional role to let the functional role inherit its privileges.
- **Read access** normally needs USAGE on the database and schema, SELECT on the table or view, and access to a warehouse. Parent-object USAGE does not grant SELECT on child tables.
- **Primary and secondary roles** can combine privileges for supported operations. CREATE authorization and ownership of newly created objects are tied to the primary role, subject to future ownership grants.
- **Managed access schemas** centralize grant decisions with the schema owner and roles holding MANAGE GRANTS. Object owners in those schemas cannot independently distribute access.
- **Future grants** cover newly created objects; grants on existing objects require separate treatment. A future OWNERSHIP grant can assign a new object to a different role than its creator.
- **Database roles** organize privileges within one database and are granted to account roles for session use. Combine roles from multiple databases through an account role; replacing a role can break existing grants.
- **SHOW GRANTS TO USER** shows grants received by a user; **SHOW GRANTS ON USER** shows privileges on that user object. SECURITYADMIN manages grants; privileged roles do not universally bypass data policies.

Account & Security

SnowPro Advanced: Architect

KNOWLEDGE CHEATSHEET

SECURITY BOUNDARIES

- **Identity** — Authenticate
- **Roles** — Authorize
- **Policies** — Protect
- **Network** — Restrict

Authentication and identity lifecycle

- **SAML federation** provides sign-in through an identity provider. **SCIM** provisions users and groups; it does not perform SSO. Configure both when centralized identity lifecycle and authentication are required.
- **OAuth** delegates access through scoped tokens. **Key-pair authentication** suits service workloads without interactive browsers. Protect private keys, rotate them and use dedicated service identities with narrow roles.
- **Authentication policies** constrain supported sign-in methods and MFA behavior. Configuring SSO alone does not prove password access is disabled; inspect user configuration and authentication policy.
- **MFA** protects interactive identities. Keep tightly controlled emergency access and validate it separately from normal federated access; do not use shared administrator credentials for application pipelines.
- **LOGIN_HISTORY** exposes successful and failed attempts. Information Schema table functions help inspect recent activity; ACCOUNT_USAGE supports historical analysis with ingestion latency. Alert on unusual failures and sources.

Private connectivity and network controls

- **Private connectivity** uses AWS PrivateLink, Azure Private Link or Google Cloud Private Service Connect. Plan supported endpoint placement, routing, private DNS, clients and stage access together.
- **Network policies** restrict ingress using allowed and blocked network identifiers or IP ranges. Apply them at supported account, user or integration scopes; network policies are not role or column permissions.
- **Private SSO** requires the identity provider and Snowflake security integration to use compatible private URLs. Validate SSO and client connectivity before restricting public access.
- **SnowCD** tests endpoint reachability. **SYSTEM$ALLOWLIST_PRIVATELINK()** provides private-connectivity endpoints; account for service, storage and certificate-validation traffic in firewall and DNS design.
- **OCSP fail-open** means a client could not obtain a valid revocation response and continued according to its configuration. Investigate responder access and client policy; TLS encryption and certificate revocation checking are separate controls.

Account & Security / Architecture & Recovery

SnowPro Advanced: Architect

KNOWLEDGE CHEATSHEET

SECURITY BOUNDARIES

- **Identity** — Authenticate
- **Roles** — Authorize
- **Policies** — Protect
- **Network** — Restrict

Masking, row filtering and tokenization

- **Dynamic Data Masking** transforms a column value at query time according to policy logic. **Row access policies** determine which rows are visible. Use each at the appropriate data boundary.
- **Conditional masking** can use another column, such as an event timestamp, to decide whether to reveal a value. **Tag-based masking** applies consistent protection to classified sensitive columns.
- **CURRENT_ROLE()** tests the primary role; **IS_ROLE_IN_SESSION()** can test active role hierarchy membership. Explicitly define authorized roles rather than assuming ownership or ACCOUNTADMIN always reveals cleartext.
- **Policy administration** separates creating a policy from applying it. APPLY MASKING POLICY and APPLY ROW ACCESS POLICY are account privileges; policy ownership and object ownership have different purposes.
- **Copies preserve query-visible values**: CTAS or INSERT SELECT can persist cleartext when the writer can see cleartext. Protect the destination explicitly; source masking does not automatically attach to a new column.
- **External tokenization** stores substituted values and can detokenize through an authorized service. Stable tokens can support approved joins without exposing original identifiers; hashing alone is not a complete anonymization design.
- **Mapping tables** encode entitlements. A supported **memoizable SQL UDF** can cache reusable lookup results in policy evaluation. Test role changes, policy dependencies and query performance.

Encryption, editions and compliance

- **Encryption** protects data in transit and at rest through a key hierarchy. **Tri-Secret Secure** combines a customer-managed cloud key with Snowflake-managed key material at the account-key layer.
- **Business Critical** adds capabilities including Tri-Secret Secure, private connectivity and account failover. PHI and payment-data designs also need applicable agreements, regional support and operational controls.
- **Enterprise** adds features such as multi-cluster warehouses, extended Time Travel for permanent tables, materialized views, search optimization and fine-grained data policies.
- **Key lifecycle** includes rotation and rekeying. Replicated data is encrypted in transit and protected by the destination account; source and destination need not share a customer-managed key.

02 Architecture, sharing and recovery

Layers and data models

- **Raw > integration > presentation** separates faithful source capture, harmonization and consumer-ready data. Preserve raw history for reprocessing while protecting sensitive fields throughout the pipeline.
- **Star schemas** place measurements in fact tables and descriptive attributes in dimensions. They simplify BI aggregation and drilldown; declare grain before designing joins and measures.

Architecture & Recovery

SnowPro Advanced: Architect

KNOWLEDGE CHEATSHEET

DATA ARCHITECTURE LIFECYCLE

Model Organize → **Share** Govern → **Clone** Develop → **Replicate** Recover

- **3NF** reduces redundancy and supports integrated relational models. **Snowflake schemas** normalize dimensions further. Snowflake supports joins; choose a model from consumer needs and maintenance costs.
- **Data Vault** separates business keys in hubs, relationships in links and descriptive history in satellites. Raw and business vault layers support source traceability and evolving rules; publish marts for consumption.
- **Multi-table INSERT** can fan out one source dataset into multiple target tables, including Data Vault structures. Parallel ingestion and independent compute help separate source loading from downstream consumption.
- **Data Mesh** combines domain-owned data products, self-service infrastructure and federated governance. Independent warehouses, governed sharing and common policy standards support this operating model.
- **Constraints on standard tables** are largely informational, with NOT NULL enforced. RELY lets the optimizer trust declared relationships; maintain and test integrity before enabling optimizations such as join elimination.

Secure Data Sharing

- **Direct shares** expose selected objects to another account in the same region without copying table data. The provider controls grants; the consumer normally pays for its query warehouse.
- **Inbound shared databases** are read-only. Consumers cannot clone them or use Time Travel on them, but can query shared data into local tables when authorized; local copies incur storage and need separate governance.
- **Provider grants** include database and schema USAGE plus SELECT on the shared tables or secure views. A newly created table becomes shared only when included in the share privileges.
- **Consumer setup** needs IMPORT SHARE and CREATE DATABASE to create a database from a share. Delegate access with IMPORTED PRIVILEGES or shared database roles, as appropriate to the share.
- **Secure views** protect view definitions and reduce optimizer-based inference risks. Expose only approved columns and rows; an ordinary view is not an equivalent privacy boundary.
- **Consumer context** such as CURRENT_ACCOUNT() or INVOKER_SHARE() can tailor shared results. Test account-aware secure views with SIMULATED_DATA_SHARING_CONSUMER; avoid assuming provider role names exist for consumers.
- **Streams on shared tables** require provider-enabled change tracking and suitable retention. The stream is a consumer-side object; monitor its staleness and the provider dependency.

Architecture & Recovery

SnowPro Advanced: Architect

KNOWLEDGE CHEATSHEET

DATA ARCHITECTURE LIFECYCLE

Model Organize → **Share** Govern → **Clone** Develop → **Replicate** Recover

Listings, exchanges and clean rooms

- **Reader accounts** allow partners without their own Snowflake accounts to query shared data. The provider administers the account and pays its compute costs; isolate access and set resource monitors.
- **Listings** package shared data for discovery, access and supported monetization. **Marketplace** reaches a broader audience; a **private listing** targets named consumers. **Data Exchange** supports a controlled member community.
- **Cross-region or cross-cloud sharing** requires data availability in the consumer region. Use supported listing auto-fulfillment or replicate to a provider-controlled account in that region and share locally.
- **Regional copies** can serve multiple consumers from one replicated dataset. Choose refresh schedules from freshness needs and include transfer, storage and maintenance costs in the design.
- **Data Clean Rooms** constrain collaboration through approved analyses and query templates. Combine identity, join and output controls to protect sensitive data while enabling joint insights.
- **Sharing restrictions** protect transfers from Business Critical to lower editions. Any authorized override must be a deliberate compliance decision; removing a restriction does not anonymize data.

Cloning and environment promotion

- **Zero-copy cloning** creates an independent snapshot using shared micro-partitions within an account. Initial creation avoids a full data copy; subsequent changes and retained history can increase storage.
- **Cross-account refresh** uses replication or a data movement pipeline. Clone an eligible local replica for writable development while retaining the replica for later refreshes; cloning does not cross account boundaries.
- **Clone grants** depend on object level and COPY GRANTS support. Database or schema cloning generally preserves grants on child objects, while the new container needs deliberate ownership and access configuration.
- **Cloned automation** needs review before activation. Tasks start suspended; pipes referencing internal stages are not cloned. Validate external-stage pipes, notifications and targets before resuming ingestion.
- **Object references** in view and procedure definitions are not universally rewritten. Fully qualified names can still point at production after cloning; inspect dependencies and enforce environment-specific access.
- **CI/CD** promotes version-controlled definitions and tests, rather than blindly replacing production data. Clone a release schema and use a supported SWAP to stage an atomic name exchange; validate grants and references.

Architecture & Recovery

SnowPro Advanced: Architect

KNOWLEDGE CHEATSHEET

DATA ARCHITECTURE LIFECYCLE

Model Organize → **Share** Govern → **Clone** Develop → **Replicate** Recover

Time Travel and recovery boundaries

- **Time Travel** queries or clones retained history with AT or BEFORE. For example, CREATE TABLE restored CLONE orders AT (OFFSET => -300) creates a snapshot from five minutes earlier.
- **BEFORE(STATEMENT => query_id)** selects the state preceding a damaging statement. Restore to a separate object, validate it and use SWAP when appropriate to preserve the current damaged version.
- **UNDROP** restores a dropped object within its retention period. If a replacement now occupies the name, rename that replacement before undropping the earlier object; object identity matters as well as its name.
- **Retention** inherits through database, schema and table settings unless overridden. The account minimum can impose a floor within supported limits. Dropping a parent container changes which recovery boundary must be considered.
- **Permanent tables** can retain up to 90 days with Enterprise or higher. **Transient and temporary tables** allow at most one day and have no Fail-safe; temporary tables can shadow permanent names within a session.
- **Fail-safe** is a separate seven-day, Snowflake-managed recovery period for permanent tables after Time Travel. It is not a self-service backup. Long-term snapshots require retained copies or an explicit archival design.

Replication, failover and client redirect

- **RPO** measures acceptable data loss; **RTO** measures acceptable recovery time. Set refresh frequency and test recovery against both, including application reconnection and pipeline restart.
- **Replication groups** synchronize supported databases and objects as a consistent group. **Failover groups** add promotion and failback. An object cannot simultaneously belong to multiple replication or failover groups.
- **Secondary replicas** are read-only until promotion where failover is supported. Replication is asynchronous: an outage can lose changes since the last completed refresh.
- **Database and share replication** is available across editions; replication of other account objects and failover/failback require Business Critical or higher. Replicate privileges and security dependencies deliberately.
- **Client Redirect** gives clients a connection URL that can be redirected to the surviving account. Configure replicated data, identities, integrations and client settings before an incident; redirect alone does not replicate data.
- **External tables** are not replicated. Recreate their definitions and external-storage access separately. Test recovery of streams, tasks, external services, object references and regional storage dependencies.

Data Engineering

SnowPro Advanced: Architect

KNOWLEDGE CHEATSHEET

SOURCE TO CONSUMPTION

Ingest Files or rows → **Track** Changes → **Transform** Dependencies → **Serve** Curated data

03 Data engineering

Stages, file formats and unstructured data

- **Internal stages** include user (@~), table (@%table) and named stages. **External stages** reference cloud storage. A stage locates files; it does not make them rows in a Snowflake table.
- **PUT and GET** transfer local files to and from internal stages through supported clients such as SnowSQL and drivers. COPY INTO loads staged files into tables or unloads query results into files.
- **Storage integrations** use a cloud identity and allowed or blocked storage locations instead of embedding credentials. One integration can serve multiple compatible stages; constrain both cloud IAM and Snowflake grants.
- **File formats** can be named objects or options on stages and COPY commands. Choose parsing, compression, null and header handling deliberately; source column matching depends on the format and configuration.
- **Directory tables** catalog staged files rather than parse their contents. Use appropriate scoped or presigned URLs for file access; an internal unstructured-file stage can use SNOWFLAKE_SSE with DIRECTORY enabled.
- **Cross-cloud external storage** can be accessed without first moving it to the account cloud. Evaluate egress and repeated-read latency; colocate frequently scanned data when justified.
- **Unloading** produces UTF-8 files through COPY INTO a stage location. Select compression and encryption deliberately, and constrain permitted export destinations through storage integrations and access controls.

Bulk loading, validation and replay

- **COPY INTO table** uses a virtual warehouse. Use parallel, reasonably sized files and narrow stage paths to avoid listing huge mixed directories; batch tiny files when latency requirements allow.
- **COPY transformations** support limited projection, omission, reordering and casting. Perform complex joins and aggregations after loading; VALIDATION_MODE does not support transformed loads.
- **Load error handling** includes continuing with valid rows or skipping a file. Inspect COPY results and load history; FILES selects specific files and RETURN_FAILED_ONLY controls reporting, not which files are loaded.
- **Load metadata** helps avoid reloading files, but does not enforce business-key uniqueness. FORCE bypasses load tracking and can duplicate records; build replay and MERGE logic around stable keys.

Data Engineering

SnowPro Advanced: Architect

KNOWLEDGE CHEATSHEET

SOURCE TO CONSUMPTION

Ingest Files or rows → **Track** Changes → **Transform** Dependencies → **Serve** Curated data

- **Immutable file names** make ingestion easier to reason about. Replacing or appending to a previously processed file is not row-level CDC; stage to an intermediate table and deduplicate before publishing.
- **File metadata** such as METADATA$FILENAME and METADATA$FILE_ROW_NUMBER supports lineage and replay diagnostics. Keep source event time distinct from load time and capture a reproducible ingestion identifier.

Snowpipe and Kafka ingestion

- **Snowpipe auto-ingest** uses cloud object notifications to queue files and serverless compute to run the pipe COPY definition. Notifications identify files; they do not carry the complete row data.
- **Snowpipe REST APIs** let applications submit staged files and retrieve load reports. Persist reports with overlapping collection windows and deduplicate them; use load-history functions for reconciliation.
- **Pipe owner privileges** include container USAGE, target-table INSERT and SELECT, and appropriate stage access. Separate pipe operations from ownership and avoid administrator roles for routine ingestion.
- **Paused pipes** have limited notification retention. Monitor SYSTEM$PIPE_STATUS, COPY_HISTORY and error notifications; restore event subscriptions and explicitly recover missed files after interruptions.
- **Snowpipe Streaming** sends rows through streaming APIs without the file-stage path. Kafka connector mode determines latency, buffering and object requirements; do not apply file-based connector assumptions to streaming mode.
- **Kafka file-based mode** manages destination tables, internal staging and pipe resources. JSON and Avro converter behavior and RECORD_CONTENT/RECORD_METADATA shape belong to connector configuration.
- **Buffers and offsets** trade ingestion latency against batching overhead. Tune flush time, bytes and record count together; preserve connector offset state and plan replay rather than assuming offset resets are harmless.
- **Staged-file lifecycle** is separate from loading. Use cloud lifecycle rules or REMOVE after verifying successful ingestion and recovery needs; AUTO_INGEST does not automatically delete source files.

Change tracking and deterministic processing

- **Change tracking metadata** supports streams and the CHANGES clause. A stream records an offset over source history, rather than storing a second full copy of the table.
- **Standard streams** expose inserts and deletes; updates appear as delete/insert pairs with METADATA$ISUPDATE. METADATA$ACTION and METADATA$ROW_ID support change processing. Append-only streams focus on inserts.

Data Engineering

SnowPro Advanced: Architect

KNOWLEDGE CHEATSHEET

SOURCE TO CONSUMPTION

Ingest Files or rows → **Track** Changes → **Transform** Dependencies → **Serve** Curated data

- **Stream offsets** advance when a consuming DML transaction commits. A SELECT alone does not consume changes. Use separate streams for independent consumers and monitor STALE_AFTER and retention extensions.
- **MERGE** needs a deterministic source-to-target match. Deduplicate or aggregate source records before updates; multiple conflicting source matches can raise errors, and unmatched duplicate source rows can create duplicate inserts.
- **Explicit transactions** group DML and queries with BEGIN, COMMIT and ROLLBACK. DDL implicitly commits an active transaction; a multi-statement DDL deployment cannot be rolled back as one ordinary transaction.
- **Stored procedure transactions** must respect scope boundaries. Pair transaction start and end in the same scope and handle failures explicitly; use clones or restoration scripts for DDL recovery.

Tasks and dynamic-table pipelines

- **Tasks** execute SQL or procedures using a warehouse or serverless compute. Build dependency graphs so a join-and-aggregate task follows all required upstream tasks, rather than the one usually finishing last.
- **SYSTEM$STREAM_HAS_DATA** can gate a task so empty input avoids unnecessary warehouse execution. It can return false positives; the task body must safely handle an empty change set.
- **Schedules** need explicit time zones. UTC avoids daylight-saving lost or repeated hours. Review task state, overlap behavior, task timeout and statement timeout when runs are skipped or terminated.
- **Dynamic tables** materialize a declared query and manage refresh dependencies. TARGET_LAG is a best-effort freshness target relative to base data, not a fixed refresh interval or hard latency guarantee.
- **DOWNSTREAM** lets intermediate dynamic tables refresh when dependent tables need them. Give a terminal table a time-based target or an explicit refresh trigger; an all-DOWNSTREAM graph has no automatic trigger.
- **Incremental refresh** suits supported queries with small changes; full refresh recomputes the result. Inspect the selected refresh mode and actual refresh cost. Use streams and tasks when imperative DML or custom orchestration is required.

Procedures, APIs and application integration

- **Owner-rights procedures** delegate a constrained operation using the owning role privileges. Grant USAGE to callers without granting broad table DELETE; validate inputs and restrict the procedure logic.
- **Caller-rights procedures** use the caller privileges and session context. Choose execution rights deliberately; a UDF computes values, while a procedure can orchestrate operations and side effects.
- **External functions** invoke remote services through a supported proxy and API integration. They can reuse existing Lambda logic; consider batching, retry safety, latency and exposure of sensitive input.

Data Engineering

SnowPro Advanced: Architect

KNOWLEDGE CHEATSHEET

SOURCE TO CONSUMPTION

Ingest Files or rows → **Track** Changes → **Transform** Dependencies → **Serve** Curated data

- **External access integrations** combine allowed egress network rules and permitted secrets for supported runtimes. A network rule permits destinations; a secret stores credentials. They serve different purposes from API integrations.
- **SQL REST API** executes SQL without a driver installation. The Python connector supports synchronous and asynchronous execution and native type conversion; avoid fetching very large results into memory unnecessarily.
- **dbt** provides SQL transformations, dependency models and tests. Use version-controlled deployments and explicit data-quality checks; dbt is not inherently the source-ingestion mechanism.
- **Git repository integration** connects an allowed repository through an API integration and, for private access, an appropriate secret. Create the repository object, fetch updates and control which revision is deployed.

Semi-structured data and lake tables

- **VARIANT, OBJECT and ARRAY** retain semi-structured data for schema-on-read. Parse JSON, navigate paths such as payload:customer.name.first::STRING, and preserve case-sensitive JSON keys.
- **FLATTEN** expands nested arrays or objects; RECURSIVE explores nested structures and key paths. INFER_SCHEMA derives a candidate relational schema from supported staged file formats.
- **Typed projections** improve predictable filtering and BI use. Extract frequently filtered timestamps to typed columns when useful; mixed types and JSON nulls can reduce automatic column extraction benefits.
- **External tables** query files in external storage through metadata and virtual columns over VALUE. Refresh metadata after file changes; event-driven refresh keeps the file catalog current without importing the rows.
- **External-table partitions** align file paths or partition metadata with selective query filters. Choose automatic or user-specified partition management deliberately; an external table is not a writable native staging table.
- **External-table row access** can protect the VALUE column. Apply filtering through supported policy attachments or a governed view; virtual columns do not support directly attached row access policies.
- **Iceberg** supplies an open table format with snapshots and ACID semantics for compatible engines. External volumes provide storage access; catalog integrations connect external metadata catalogs. Metadata-file configuration depends on the catalog type.
- **Catalog ownership** determines Iceberg maintenance and feature support. With customer-managed storage, plan lifecycle, snapshot retention and recovery alongside Snowflake; distinguish open-format tables from ordinary external tables.

Data Engineering / Performance & Operations

SnowPro Advanced: Architect

KNOWLEDGE CHEATSHEET

SOURCE TO CONSUMPTION

Ingest Files or rows → **Track** Changes → **Transform** Dependencies → **Serve** Curated data

Snowpark, applications and model serving

- **Snowpark DataFrames** express transformations that execute in Snowflake. Push work to the platform rather than extracting an entire large dataset into a client process.
- **Snowpark-optimized warehouses** provide memory-rich compute for suitable workloads. Limit competing queries when one procedure needs most resources; multi-cluster scaling does not add memory to one execution.
- **Streamlit** builds Python applications and interactive views for business users. Keep data authorization, application execution rights and secret handling aligned with the intended audience.
- **Snowpark Container Services** runs containerized workloads on compute pools, including supported GPU configurations for training and inference. Service endpoints support application access to deployed models; choose pools from runtime and capacity needs.
- **Malformed input** needs quarantine and observable failures. Use safe parsing such as TRY_PARSE_JSON, procedure exception handling and event-table telemetry; validate records before publishing curated tables.

04 Performance and operations

Warehouse sizing and concurrency

- **Scale up** increases resources available to a query. **Scale out** adds warehouse clusters for concurrent queries. Separate load, transformation and BI warehouses to isolate contention and spending.
- **Multi-cluster auto-scale** adjusts cluster count between configured bounds. Maximized mode keeps the specified clusters running; choose it only when predictable sustained concurrency justifies the cost.
- **Standard scaling policy** favors responsiveness by starting capacity sooner; **Economy** favors fuller clusters and can tolerate more queuing. Use warehouse load history to evaluate the trade-off.
- **MAX_CONCURRENCY_LEVEL** limits simultaneous statements per cluster. Lowering it can give each heavy query more resources; increasing it may reduce queues for small queries but can increase memory contention.
- **Auto-suspend and auto-resume** reduce idle compute. Very short suspension intervals lose warm warehouse cache and can incur repeated resume minimums; balance idle credit cost against response-time needs.

Performance & Operations

SnowPro Advanced: Architect

KNOWLEDGE CHEATSHEET

DIAGNOSE BEFORE SCALING

- **Queues** Concurrency
- **Spill** Memory
- **Scan** Pruning
- **Spend** Metering

Query Profile and bottleneck diagnosis

- **QUERY_HISTORY** separates compilation, execution, queue and blocking time. High compilation time can signal complex generated SQL; adding warehouse resources does not directly solve every planning bottleneck.
- **Bytes spilled to local or remote storage** indicate memory pressure in operations such as joins and aggregation. Reduce intermediate rows, split processing, reduce competing queries or increase warehouse size.
- **Join explosion** occurs when a join outputs far more rows than it consumes. Check join predicates, duplicate keys and intended grain before treating compute size as the solution.
- **Partitions scanned versus total** reveals pruning effectiveness. Review filter selectivity, clustering and expressions; a query that reads most rows may gain little from selective access structures.
- **Warehouse load history** distinguishes running work from overload and provisioning queues. QUERY_HISTORY and LOCK_WAIT_HISTORY help identify blockers; use query hashes to group recurring expensive patterns.

Micro-partitions and clustering keys

- **Micro-partition metadata** tracks column ranges and other statistics so Snowflake can prune irrelevant data. Native tables are automatically partitioned; manual date-sharded tables are usually unnecessary.
- **Clustering keys** suit large tables repeatedly filtered or joined on selective columns. Choose a small set tied to real access patterns; projected columns alone are not a reason to cluster.
- **Cardinality** must support useful partition discrimination without excessive maintenance. Reduce extreme timestamp cardinality with an order-preserving date expression; multi-column keys generally place lower-cardinality expressions first.
- **SYSTEM$CLUSTERING_INFORMATION** and **SYSTEM$CLUSTERING_DEPTH** describe overlap and depth for chosen expressions. Lower overlap generally helps pruning; a high depth is not evidence of excellent clustering.
- **Automatic Clustering** rewrites micro-partitions and consumes serverless credits. Replaced partitions can remain in Time Travel and Fail-safe, increasing retained storage; measure query savings against maintenance cost.

Performance & Operations

SnowPro Advanced: Architect

KNOWLEDGE CHEATSHEET

DIAGNOSE BEFORE SCALING

- **Queues** — Concurrency
- **Spill** — Memory
- **Scan** — Pruning
- **Spend** — Metering

Search, materialization and acceleration

- **Search optimization** maintains access paths for supported highly selective lookups and predicates. It can help different teams search different columns; ordinary users need SELECT, while setup needs table OWNERSHIP and schema ADD SEARCH OPTIMIZATION.
- **Search configuration** can target columns and operations with ON. Explicit ON configuration does not automatically cover later columns; inspect DESCRIBE SEARCH OPTIMIZATION and maintenance progress.
- **Replicated search optimization** carries the table setting to the secondary database. Search access paths are rebuilt there automatically and incur destination maintenance and storage costs.
- **Materialized views** persist eligible single-table results and are maintained automatically. They have restrictions such as no joins or nested subqueries; supported aggregates include MIN and MAX.
- **Alternate clustering** on a materialized view can serve an access pattern poorly served by the base key. For complex joins or periodic aggregates, use suitable dynamic tables or task-maintained summary tables.
- **Query acceleration service** offloads eligible portions of queries to serverless resources. Estimate eligibility and cost; it does not accelerate every operator or replace concurrency isolation.
- **Approximate aggregation** trades exactness for speed and memory savings. HLL or APPROX_COUNT_DISTINCT estimates cardinality; MinHash estimates set similarity, and APPROX_TOP_K estimates frequent values.

Result reuse and warehouse cache

- **Persisted query results** are held separately from a warehouse and can be reused across warehouses when query, data, configuration and authorization requirements match. They do not require a warm warehouse data cache.
- **Result retention** normally starts at 24 hours; qualifying reuse resets that interval up to a 31-day limit from initial execution. Result reuse is an optimization, not a durable archive.
- **Warehouse cache** holds previously read data locally. Suspending a warehouse drops that cache; resuming it does not preserve the benefit of a morning warm-up.
- **Reuse eligibility** can be affected by table changes, query text and non-reusable functions. Time-dependent expressions can interfere with reuse; use stable query inputs and measure actual behavior.

Performance & Operations

SnowPro Advanced: Architect

KNOWLEDGE CHEATSHEET

DIAGNOSE BEFORE SCALING

- **Queues** — Concurrency
- **Spill** — Memory
- **Scan** — Pruning
- **Spend** — Metering

Monitoring, alerts and cost controls

- **ACCOUNT_USAGE** supports historical account reporting; **ORGANIZATION_USAGE** aggregates supported organization metrics. Information Schema functions often provide fresher operational data within shorter windows.
- **Resource monitors** track warehouse credits and can notify, suspend after running statements, or suspend immediately according to configured triggers. Reaching a quota alone does not define an action.
- **Account and warehouse monitors** apply together. An account-wide suspend trigger can affect all applicable warehouses even when individual quotas remain; monitors do not cap every serverless service.
- **Budgets and alerts** complement resource monitors for supported object and serverless costs. Monitor warehouse metering, storage, transfer, Snowpipe, task, clustering, materialized-view and search-optimization usage.
- **Snowflake alerts** evaluate conditions and run configured actions. Use notification integrations for delivery and event tables for application logs, traces and runtime errors; select a metric source appropriate to the incident.
- **Timeouts** constrain runaway statements and task runs at supported scopes. Query attribution and metering are better cost evidence than simply multiplying overlapping query durations by the warehouse credit rate.