**Snowflake Interview Questions**

**Snowflake features** – zero copy cloning, time-travel, fail-safe, handles semi-structured data.

**Architecture of snowflake** – multi-cluster shared architecture

**Types of cache** –

result cache(24hrs),

local disk or warehouse cache (suspends or scales down),

metadata cache (table structure changes), remote disk cache.

**Layers:**

Cloud services layer – metadata management, authentication, user access control.

Compute layer (virtual warehouse layer) – query execution and data processing, query cache

Storage layer – micro-partitioning, time-travel, fail-safe, zero-copy cloning.

**Editions:**

Standard, enterprise, business critical, Virtual private snowflake.

**Virtual warehouse**:

compute engine, runs queries and processes data, scaling up/down, sizes, paused when not in use.

When not in use:

query execution stops

warehouse cache is cleared

no compute cost

**Micro-partitioning:**

automatic, small storage units - divides data into for faster querying.

No need to manually define partitions.

Each micro-partition contains **50MB–500MB** of data

Helps in automatic clustering and query optimization**.**

**Time Travel:**

1-90 days depending on snowflake edition

query, restore, or clone historical data.

accidental deletions and auditing

Uses the UNDROP, SELECT AT, and CLONE commands.

**Fail-safe:** 7-day additional back-up for disaster recovery, extra layer of protection beyond time-travel.

**Snowpipe:**

Continuous data ingestion service - which automatically loads new data from cloud storage (S3, Azure Blob, GCS) into Snowflake as soon as it arrives.

event-driven architecture to avoid batch loads.

Loads only new files, reducing costs.

Supports semi-structured data (JSON, Parquet, Avro, etc.)

**Streams and Tasks:**

Tracks changes (INSERTS, UPDATES, DELETES) on a table.

Used for Change Data Capture (CDC).

Helps automate ETL processes.

Works with Tasks & Stored Procedures to process changes.

**Is\_update** and **meta$data\_action** are the 2 columns which indicates CDC updates

Tasks: scheduling

**Secure Views:**

read-only views that hide sensitive columns from unauthorized users.

Prevents users from accessing the underlying query logic.

**Types of tables:**

Permanent - default

Temporary - Exist only for the session and get deleted automatically.

Transient - Persist across sessions but do not support Time Travel or Fail-Safe.

External - Reference data stored in S3, Azure, GCS, without actually loading it into Snowflake.

**Schema types:**

Public schema: default

User-defined schema: custom made by user

Information schema: system-generated schema

Temporary schema: only for session duration

**Clustering:**

Automatic Micro-Partitioning

Manual Clustering using Cluster Keys improves query performance for large tables.

Avoids full table scans by reducing partition pruning.

**Stages:**

Internal stage: Stores files within Snowflake

External stage: Connects to S3, Azure Blob, or Google Cloud Storage.

User stage: Each Snowflake user gets a private storage area

Table stage: Each table has an automatic storage location.

**Load data into snowflake:**

Copy into: Bulk Load (100mb – 5gb per file)

Snowpipe: continuous data ingestion (<100mb per file)

Insert into: Small Data Inserts (few mb’s per query)

PUT + COPY INTO: staging large files before loading (upto 5gb per file)

**Secure Views:**

Prevent unauthorized users from viewing underlying query logic.

Used to protect sensitive data while sharing results.

**Materialized Views:**

store precomputed query results for faster performance.

they consume storage but improve query speed.

Snowflake refreshes this view **automatically** when new data arrives.

**Optimize query performance:**

Use Clustering Keys for large tables.

Use Materialized Views for precomputed aggregations

Optimize Warehouse Sizing to avoid under/over-provisioning.

Use Query Result Caching to reuse previous results.

**SHOW → Lists objects like tables, views, schemas, roles**

**DESCRIBE → Provides detailed metadata about a specific object.**

**Alter command**:

Used to modify existing tables (add/remove columns, rename table, change clustering keys).

**Query history:** check query execution.

**Sharing data between snowflake accounts:** secure data sharing.

Flow of data ingestion from aws s3 to snowflake steps

Can we reload the same table twice into snowflake

When a query is taking long time to run, how to make query run faster

What is scale-up and scale-down

Stored procedure syntax

Command to get back the deleted table records for 1 day- undrop command

Scenario based questions:

**Data Ingestion:**

1. You have a large dataset (5 TB) stored in S3. How would you efficiently load this data into Snowflake?

* Use **COPY INTO** for bulk ingestion, breaking data into **100-250MB files** for optimal performance.
* Use **Snowpipe** for continuous streaming of data.
* Optimize **file formats** (e.g., Parquet is better than CSV for efficiency)

1. Your company is migrating petabytes of data from S3 to Snowflake. How do you handle such a large migration efficiently?

* Use partitioning
* Use parallel ingestion
* Optimize warehouse usage
* Leverage Snowpipe for real-time ingestion

1. You need to migrate JSON data from S3 to Snowflake. How will you handle nested JSON structures?

* Use **VARIANT** data type in Snowflake.
* Load JSON data into a **staging table** and then parse it using FLATTEN().

1. You need to load data from S3 into multiple Snowflake tables based on file name patterns. How do you handle this?

* Use **COPY INTO** with file patterns to direct data into appropriate tables.
* Alternatively, use **Snowflake Tasks** to process files after loading

1. Your ingestion pipeline must support both batch and real-time data from S3. How do you design it?

* Use **COPY INTO** for batch processing.
* Use **Snowpipe** for real-time ingestion.
* Implement **stream and task automation** for transformation.

S3 → Snowpipe (real-time) → Staging Table → Streams & Tasks → Processed Table

**Duplication:**

**Distinct, row\_number, merge for deduplication.**

**De-duplication:** removing duplicate records to avoid redundancy.

1. What happens if a file is loaded twice into Snowflake? How do you prevent duplicate records?

* **Duplicates can occur** if the same file is loaded multiple times.
* Use **metadata tracking** (LAST\_MODIFIED, FILENAME).
* Use MERGE INTO instead of INSERT to update existing records.

1. You need to remove duplicate records before inserting data from S3 into Snowflake. How do you handle this?

* Load raw data into a **staging table**.
* Use MERGE INTO to remove duplicates before inserting into the final table.

**Snowpipe & Automation Scenarios:**

1. You are working with real-time streaming data from S3 to Snowflake. How would you set up Snowpipe?

* **Create an S3 storage integration** and define an **external stage**.
* **Create a Snowpipe** to automatically ingest files.
* **Configure S3 Event Notifications** to trigger Snowpipe.

1. How would you monitor Snowpipe failures?

* Check **Snowflake COPY\_HISTORY**
* Use **AWS CloudWatch** to monitor S3 event notifications.
* Implement **error logging** in Snowflake tasks.

1. Your Snowpipe is consuming high credits due to frequent small file loads. How do you optimize costs?

* **Batch files before ingestion** instead of sending small files frequently.
* **Use file arrival triggers** (e.g., AWS Lambda) to group files before Snowpipe ingestion.
* **Use larger files (100-250MB each)** for optimal cost efficiency.

1. Your S3 files are ingested into Snowflake using Snowpipe, but you notice a delay in data availability. What could be the issue?

* **Check if Snowpipe is running**
* **Check AWS S3 event notifications**.
* **Check Snowflake’s COPY\_HISTORY** for ingestion errors.

1. You are using Snowpipe, but some data is missing. How do you troubleshoot?

* Verify **S3 event notifications** are properly configured.
* Check **PIPE\_USAGE\_HISTORY**:
* Ensure that files were not **already loaded and skipped**.

**Performance Optimization Scenarios:**

1. A large COPY INTO query is running slowly. How would you improve its performance?

* Ensure **small file sizes** (100-250MB) instead of large files.
* Use **parallel execution** with Snowflake’s multi-cluster warehouse.
* Convert CSV files into **Parquet** to reduce file size and improve query efficiency.
* Increase **warehouse size** (scale-up) for faster ingestion.
* **Use partition pruning** to reduce scanned data volume.

1. After loading data, queries are slow. How do you optimize Snowflake performance?

* Use clustering keys
* Create **materialized views** to precompute results.
* Optimize **micro-partition pruning** by using **sorted data**.

1. A COPY INTO command is failing intermittently. What steps would you take to debug?

* Check **error logs**
* Validate **file format compatibility**.
* Ensure **correct file permissions in S3**.

1. Your S3 ingestion pipeline is consuming too many Snowflake credits. How do you optimize costs?

* **Optimize warehouse size & suspend inactive warehouses**
* **Use Parquet instead of CSV** to reduce data storage and improve query performance.
* **Batch smaller files** together to avoid excessive Snowpipe invocations.

**Error Handling & Debugging Scenarios & Recovery:**

1. You loaded a file from S3, but some records are missing in Snowflake. How do you debug this?

* Check **COPY\_HISTORY** for errors

(**SELECT \* FROM TABLE(INFORMATION\_SCHEMA.LOAD\_HISTORY());)**

* Verify **file formats and delimiters** (CSV headers, JSON structure).
* Ensure **correct file permissions in S3**.

1. You are using Snowpipe, but some data is missing. How do you troubleshoot?

* Verify **S3 event notifications** are properly configured.
* Check **PIPE\_USAGE\_HISTORY**:
* Ensure that files were not **already loaded and skipped**.

1. You accidentally deleted a table that was ingested from S3. How do you recover it?

* Use **Time Travel** to restore.
* UNDROP command – time, day (undrop table table\_name)
* If **Time Travel has expired**, use **Fail-Safe (Snowflake support required).**

1. After ingestion, some records contain incorrect values. How do you trace and fix the issue?

* Compare ingested data with source S3 data.
* Check **file encoding issues** (e.g., UTF-8 vs. ANSI)
* Use Snowflake’s **QUERY\_HISTORY** to trace the COPY INTO command.

1. Your COPY INTO command failed due to corrupted files in S3. How do you handle this?

* Check error details
* **Use VALIDATION\_MODE** to test file integrity before loading
* **Exclude problematic files** using metadata filters.

1. How do you ensure that a failed data load does not insert partial records into Snowflake?

* Use **transactions** to ensure atomicity.
* load data into a **staging table first** and then MERGE INTO the main table.

1. The ingested data contains null values and incorrect date formats. How do you clean it in Snowflake?

* Use **IS NULL checks** and transformations
* Create a **stored procedure** to clean data before inserting it into the final table.

1. How do you handle schema evolution when loading JSON files from S3 into Snowflake?

* Use **VARIANT type** to handle dynamic schema
* Use FLATTEN() to extract JSON fields dynamically
* If schema changes frequently, implement a **dynamic column addition** mechanism.

**Incremental loading and Data Transformation & Processing Scenarios**

1. How would you process ingested data before inserting it into the final Snowflake table?

* Use **Streams & Tasks** for incremental processing.
* **Stream** to track changes
* **Task** to process data

1. How do you ensure data consistency when loading incremental data from S3 to Snowflake?

* Use **metadata columns (LAST\_MODIFIED)** to track updates.
* Implement **Change Data Capture (CDC)** with **streams and tasks**.
* Use MERGE INTO instead of INSERT to handle updates.

1. You need to remove duplicate records before inserting data from S3 into Snowflake. How do you handle this?

* Load raw data into a **staging table**.
* Use MERGE INTO to remove duplicates before inserting into the final table.

1. You need to track daily incremental changes from S3 data and apply them to Snowflake. How would you do this?

* Use **Streams and Tasks** to track changes and apply updates
* Create a **stream on the staging table** to capture new records
* Use a **task to merge changes into the final table**:

**Data Recovery & Security:**

1. A table was accidentally deleted. How do you recover it?

* Use **Time Travel (UNDROP TABLE my\_table;)**
* If **Time Travel period expired**, use **Fail-Safe (7 days)**

1. You need to implement a rollback mechanism if an incorrect dataset is ingested from S3. How do you do this?

* **Use Time Travel** to restore data
* If Time Travel is unavailable, reload data from S3 with **corrected transformations**.

1. How do you secure S3 data before loading it into Snowflake?

* **Encrypt data** at rest in S3 (SSE-S3, SSE-KMS).
* Use **IAM roles** instead of access keys.
* Restrict **Snowflake access** using **RBAC (Role-Based Access Control)**.

1. You need to allow only specific users to load data from S3 to Snowflake. How do you implement this?

* Use **ROLE-BASED ACCESS CONTROL (RBAC)**.
* Grant access only to authorized roles.
* Configure **IAM role-based authentication** instead of hardcoded credentials.

1. How do you encrypt and secure data while migrating from S3 to Snowflake?

* Use S3 encryption.
* **Restrict access to Snowflake’s storage integration**.
* Enable **network policies** to restrict data access.

1. How do you ensure that sensitive data remains encrypted during migration?

* **Enable S3 encryption** (SSE-S3, SSE-KMS)
* **Use Snowflake's column-level encryption** for sensitive data.

**Security and Grant Access:**

1. A user should be able to query a table but not modify it. How do you grant the required permissions?

* *GRANT SELECT ON my\_table TO user1;*
* *GRANT USAGE ON SCHEMA my\_schema TO user1;*
* *GRANT SELECT ON ALL TABLES IN SCHEMA my\_schema TO user1;*

**Data Sharing:**

1. Your company wants to share a dataset with a third party without creating a copy. How do you achieve this in Snowflake?

* Use secure data sharing
  + Create a **share**
    - CREATE SHARE my\_share
  + Add a **table/view** to the share
    - GRANT USAGE ON DATABASE my\_db TO SHARE my\_share;
    - GRANT SELECT ON TABLE my\_table TO SHARE my\_share;
  + Grant access to the consumer
    - GRANT USAGE ON SHARE my\_share TO ACCOUNT consumer\_account;

**Advanced Scenarios:**

**File Formats:**

1. Can you load multiple file formats (CSV, JSON, Parquet) into the same Snowflake table?

* Yes, but use **different file formats** in COPY INTO.
* Use **variant data type** for semi-structured data.

1. The S3 bucket contains multiple file formats (CSV, JSON, Parquet). How would you load them into Snowflake dynamically?

* Create a **separate file format** for each data type.
* Use **auto-detection** of file format using the FILE\_FORMAT parameter.

Miscellaneous:

1. You need to migrate a 10-year-old historical dataset from S3 to Snowflake. What approach would you take?

* **Divide the dataset into smaller chunks** (e.g., **year-wise partitions**).
* **Use parallel COPY INTO** commands to optimize ingestion.
* Archive older data in **separate Snowflake tables** for faster query performance.

1. You need to migrate data from S3 to Snowflake and then archive old data. What’s the best approach?

* Load new data into Snowflake.
* Archive old data into an **external table** linked to S3

1. You need to archive older data from Snowflake back to S3 for cost optimization. How do you do this?

* Export data using COPY INTO
* Set up **external tables** in Snowflake to query archived data without reloading.

1. Your S3 files are ingested into Snowflake using Snowpipe, but you notice a delay in data availability. What could be the issue?

* Check if Snowpipe is running
* **Check AWS S3 event notifications**.
* **Check Snowflake’s COPY\_HISTORY** for ingestion errors.

1. Your ingestion pipeline needs to handle schema evolution dynamically. How do you achieve this?

* Use **Snowflake’s VARIANT type** to store JSON with flexible schema.
* Use **Schema Detection** (AUTOMATICALLY DETECT SCHEMA in Snowflake).

1. How would you design an ETL pipeline for an end-to-end data migration from S3 to Snowflake?

**Extract**: Use COPY INTO or Snowpipe to ingest data.

**Transform**: Use **Streams & Tasks** for data processing.

**Load**: Store cleaned data into a **final table**.

**Monitor**: Use **query history and metadata tracking**

**Optimize: Implement clustering keys & materialized views.**

1. You need to migrate a large dataset from S3 but minimize downtime for ongoing analytics. What’s your strategy?

* **Migrate in incremental batches** instead of one-time bulk load
* **Use streams to track changes** and update only new records.
* **Use zero-copy cloning** to minimize impact on active queries.

1. Your company is merging multiple S3 datasets into a single Snowflake table. How do you avoid key conflicts?

* Use a **composite primary key** (e.g., customer\_id + source\_system).
* Implement **UUIDs or hash-based unique keys** to prevent duplicates.

1. Your JSON data has a nested address field ({"city": "New York", "zip": "10001"}). How do you extract it in Snowflake?

*SELECT data:address.city::STRING AS city,*

*data:address.zip::STRING AS zip*

*FROM json\_table;*

1. You notice frequent full table scans. How do you improve query performance?

* Use **Clustered Tables**
* Use **Materialized Views**

Migrating data from oracle to snowflake:

1. Aws end: aws s3, iam role
2. Snowflake end: warehouse, database, schema
3. In aws s3 we have data, or internal (using python boto3 library)
4. Create file format
5. Create integration object (provider, role\_arn, location)
6. Desc integration integration\_object
7. Based on integration\_object we update the trust policy in AWS IAM.
8. Create external stage(integration\_object, file\_format)
9. Check for tables
10. Create snowpipe (stage, file\_name, file\_format)( (to load raw data from S3 to a Snowflake table automatically)
11. Show pipes
12. System$pipe\_status(‘table\_name’)
13. Create stream on the table/view (materialized view (or) dynamic view)( to track changes in the table)
14. Create task to process stream data (to process new changes from the stream into another table)
15. Enable task
16. Check task status (show tasks)
17. Create AWS SNS topic
18. Subscribe to the topic
19. Create snowflake notification integration
20. Use task to send notification
21. DBT
22. Providing privileges on warehouse, database, schema, tables
23. Dbt run
24. Dbt run –model\_name: creates a new db in snowflake which is orchestrated
25. Can be used for analysis

Day-to-day:

Once we log-in, go to jira board, where tasks will be assigned

Take task, differ every single day.. mostly data\_ingestion, do transformations in dbt

Scrum-call, sprint of 4 weeks

Run the scripts.. change it accordingly

Such as file\_formats, check for privileges of tables

Check pipe\_status

Once the data is ingested, check for data\_quality