Storage Internals & Micro-Partitions

What are Micro-partitions?

🔍 **Industry Problem** Modern data-driven enterprises like **Netflix**, **Amazon**, or **Meta** generate petabytes of data every day—user logs, transactions, content interactions, personalization data, etc.

Traditional row-based and even columnar databases often struggle with:

- **Managing massive volumes** of data efficiently
- **Fast query response** on large tables (billion+ rows)
- Poor partitioning logic needing **manual tuning**
- Expensive **storage scanning** due to full-table scans

✅ Snowflake's Solution: Micro-partitions

Snowflake **breaks down every table into hundreds to millions of **immutable, columnar, compressed** files called **micro-partitions**.

🔑 Key Characteristics

Feature	Description
Size	Each micro-partition stores ~16 MB (compressed) of data
Columnar	Each column is stored independently, enabling selective reads
Automatic	You don't define partitions—Snowflake handles it automatically
Metadata-rich	Each micro-partition stores min/max ranges, null counts, and value lists per column
Immutable	Once written, a micro-partition doesn't change—new changes create new partitions

🔄 How it Works (Behind the Scenes)

- When you insert 10 million rows into a table, Snowflake:
    - **Splits it into many micro-partitions**
    - **Compresses and organizes** them by column
    - **Stores min/max and statistical metadata**
    - Makes them **available for pruning**

🏢 Real Industry Example – Netflix (Content Analytics)

Netflix collects **video watch logs** from every device (TVs, phones, browsers) for personalization, recommendation, A/B testing, etc.

Each event:

`user_id`, `video_id`, `device_type`, `play_time`, `watch_duration`, `region`, `timestamp`

They use Snowflake to:

- Store billions of rows per day
- Organize them automatically into **micro-partitions**
- Enable filtering by region/date/user without full scans

So, a query like:

```
SELECT COUNT(*)
FROM VIDEO_LOGS
WHERE REGION = 'IN' AND DATE(timestamp) >= CURRENT_DATE - 1;
```

Only scans partitions that match the date and region (instead of the entire table)

Columnar Storage Format & Compression

Industry Problem

At scale (think **Google Search logs**, **Amazon transaction data**, or **Meta ad clicks**):

- Reading entire rows (all columns) is extremely inefficient
- Uncompressed storage eats up huge cost and IO
- Legacy systems read 90% irrelevant data

✅ Snowflake's Solution: Columnar + Compression

Snowflake's storage engine is:

- **Fully columnar**: Data is stored **by column**, not row
- **Heavily compressed**: Applies best-fit compression per column
    - Run-Length Encoding, Dictionary, Delta, Bit-Packing, etc.

🚀 Benefits:

- Only **columns used in SELECT** are read from disk
- Compression ratios of **5x to 15x** are common
- Read and write optimized for analytical queries

🧠 Example: How It Helps Meta (Facebook Ads Team)

Meta stores **ad impressions and clicks** with fields:

`user_id`, `ad_id`, `campaign_id`, `click_time`, `device_type`, `impression_cost`

A marketing analyst querying **daily click-through rates** only needs:

```
SELECT campaign_id, COUNT(*)
FROM ADS_EVENTS
WHERE click_time BETWEEN '2024-01-01' AND '2024-01-02'
GROUP BY campaign_id;
```

Snowflake only reads:

- `campaign_id`
- `click_time`

Result: Massive reduction in **I/O and query latency** despite billions of rows

Automatic Creation & Immutability of Micro-partitions

🔍 Industry Problem / Real-World Challenge

In legacy warehouse solutions (e.g., Hadoop Hive or on-prem Teradata):

- You **must predefine** partitioning keys
- Poor choice leads to **query inefficiency**
- Data **mutations** rewrite entire partitions
- **Concurrency and time-travel are hard**

✅ Snowflake's Solution: Auto & Immutable Micro-partitions

📌 **Automatic Creation:**

- Snowflake **automatically partitions** data as it's loaded
- Based on **natural ingestion order** (no user intervention)
- Adds **metadata** to aid later pruning

📌 **Immutability:**

- Once a micro-partition is written, it is **never updated**
- Updates/deletes result in **new partitions**
- Ensures:
    - Fast concurrency
    - Snapshot isolation (Time Travel)

    - Audit safety and compliance

🏢 Real-World Example: Amazon Order Data

In Amazon's ecommerce systems:

- Every order has: `order_id`, `customer_id`, `order_date`, `order_amount`, `status`

Their BI systems might:

- Load millions of orders hourly
- Occasionally correct order statuses or refunds

Snowflake allows:

- Automatic ingestion → partitions created
- Refund updates? → Old partitions closed, **new partition created**
- Analysts still see **latest view**, but older partitions exist for:
    - **Time Travel**
    - **Audit trails**
    - **Reversibility**

🔄 Example: Observing Partition Evolution

```
-- View table metadata
SELECT *
FROM INFORMATION_SCHEMA.TABLE_STORAGE_METRICS
WHERE TABLE_NAME = 'ORDERS';

-- View clustering statistics
SELECT SYSTEM$CLUSTERING_INFORMATION('ORDERS');
```

- You'll observe micro-partition counts and last update times changing
- But no partition is modified—new data = new partitions

📚 Summary

Subtopic	Key Idea	Benefit	Industry Example
What are Micro-partitions?	Snowflake breaks large data into small, columnar blocks	Enables automatic scalability and parallelism	Netflix watch logs
Columnar + Compression	Store columns separately with compression	Only required data read; faster queries, less storage	Meta ad clicks
Auto + Immutable Partitions	Snowflake handles everything + never updates partitions	Enables time travel, ACID compliance, and high concurrency	Amazon order pipeline