Metadata & Query Pruning in Snowflake

Why This Topic Matters (FinTech Lens)

In a FinTech company, you often deal with:

- **High-volume** transactional data (e.g., payments, trades, customer activity)
- Regulatory need for **query transparency**, **speed**, and **efficiency**
- Complex filtering (e.g., get trades for a specific customer between 2 dates)

In traditional systems:

- Indexes are manually maintained
- Statistics need regular updates
- Poor filtering = full table scans = ⏳ slow queries

🧠 Subtopic A: Metadata Store & Pruning Mechanism

🔍 The Industry Problem

Imagine you're storing 3 billion payment transactions in your Postgres, MySQL, or Hive-based system. Querying:

```
SELECT * FROM transactions WHERE transaction_date = '2024-01-01';
```

⚠️ Issues:

- Full table scan (if not indexed)
- Indexes = high maintenance
- Updates → statistics become stale
- Manual partitioning logic needed

✅ Snowflake's Solution: Rich Metadata & Automatic Pruning

Snowflake stores **detailed metadata at the micro-partition level**, such as:

- `min` / `max` values for each column
- `null` counts
- Distinct values (when feasible)

Each **micro-partition** (≈16MB) has its own metadata block.

🔍 Pruning: Selective Scan Optimization

When you run a query:

```
SELECT *
FROM PAYMENT_TXNS
WHERE TRANSACTION_DATE = '2024-01-01';
```

Snowflake:

1. Scans metadata first
2. Skips micro-partitions where `TRANSACTION_DATE` ≠ '2024-01-01'
3. Only reads relevant partitions — this is called **pruning**

🔧 Example (FinTech - Stripe or Razorpay style)

A FinTech company stores payment data:

transaction_id	user_id	amount	status	transaction_date
T123456	U001	100.00	SUCCESS	2024-01-01
T123457	U002	200.00	FAILED	2024-01-02
...	...	...	...	...

When analysts filter for a specific date or user:

```
SELECT COUNT(*)
FROM PAYMENT_TXNS
WHERE transaction_date BETWEEN '2024-01-01' AND '2024-01-07';
```

✅ Snowflake skips 90%+ of the table if most partitions are outside that range.

🧠 Subtopic B: SYSTEM$CLUSTERING_INFORMATION – Understanding Data Organization

🔍 The Industry Problem

After weeks/months of use, your large Snowflake table starts growing. Over time:

- Rows get added/updated/deleted
- Original “natural clustering” becomes fragmented
- Filtering becomes slower

✅ Snowflake’s Answer: `SYSTEM$CLUSTERING_INFORMATION()`

This **table function** helps you analyze how well your data is organized **based on one or more columns**.

It returns:

- **average_depth**: How many overlapping partitions exist for a filter condition
- **partition_count**: Total partitions considered
- **total_overlaps**: How many times partitions overlap on clustering keys

💡 How It Works (Simplified)

When you cluster by `transaction_date`, you want all rows for a day close together.

Over time:

- Inserts from various regions/systems disrupt the order
- You can **measure** how "scattered" the values are

```
SELECT SYSTEM$CLUSTERING_INFORMATION('PAYMENT_TXNS');
```

Or, for advanced usage:

```
SELECT SYSTEM$CLUSTERING_INFORMATION(
  'PAYMENT_TXNS',
  '(transaction_date)'
);
```

➡️ If **depth** is high, consider **re-clustering**.

🏢 Real Example: PayPal Risk Analytics

The risk team clusters data on `customer_id` and `risk_score`.

Over time, clustering decays due to ongoing updates.

They monitor clustering info weekly:

- If `total_overlaps` becomes too high, a **manual re-cluster** is triggered
- Keeps risk dashboard queries fast and cost-effective

🧠 Subtopic C: Query History & Pruning Effect

🔍 The Industry Problem

You optimize queries but want to validate:

- Are micro-partitions actually being pruned?
- What's the impact of changing filters?

✅ Snowflake's Visual & Metadata Tools

1. Snowsight Query History

Go to **Snowsight** → **History** → **Click a Query** → **Profile Tab**

You'll see:

Field	Meaning
Partitions scanned	How many partitions were read
Partitions pruned	How many were skipped (👍 good!)
Bytes scanned	Actual disk read
Bytes billed	What you are charged for

2. Example Analysis

```
-- Bad filter (can't prune)
SELECT * FROM PAYMENT_TXNS
WHERE TO_CHAR(transaction_date, 'YYYY-MM-DD') = '2024-01-01';

-- Good filter (prunes partitions)
SELECT * FROM PAYMENT_TXNS
WHERE transaction_date = '2024-01-01';
```

🔍 Use the query profile to show participants the difference — fewer partitions read, faster query, less cost.

🔚 Summary (Cheat Sheet)

Subtopic	Problem it Solves	How Snowflake Helps	FinTech Use Case
Metadata & Pruning	Full scans, slow filters, costly queries	Metadata-rich micro-partitions = fast pruning	Transaction queries by date/user
SYSTEM$CLUSTERING_INFORMATION	Query performance degrades over time	Exposes clustering depth/quality	Reorganize risk/customer data views
Query History & Profile	Hard to validate query efficiency	Visual analysis of scanned vs pruned partitions	Audit fraud queries or slow BI views