**Scenario 1: Table without a Cluster Key**

**1. Create Table:**

SQL

CREATE TABLE my\_large\_table (

id INT,

product\_category VARCHAR,

order\_date DATE

);

* **What happens:** Snowflake creates the logical structure for the my\_large\_table. At this point, no storage has been consumed because there is no data.

**2. Insert Data:**

Now, let's insert some data. For this example, imagine we are inserting data from multiple sources in a random order, not sorted by date or category.

SQL

INSERT INTO my\_large\_table (id, product\_category, order\_date) VALUES

(1, 'electronics', '2024-03-01'),

(2, 'books', '2024-01-15'),

(3, 'electronics', '2024-02-20'),

(4, 'clothing', '2024-01-05'),

(5, 'books', '2024-03-25');

* **What happens:** Snowflake takes the incoming data and organizes it into **micro-partitions**. Because there is no cluster key, the data is simply added to the next available micro-partition in the order it was received. This is called **natural clustering**.
* **Result:** The data for 'books' and 'electronics' gets scattered across multiple micro-partitions. When you run a query like SELECT \* FROM my\_large\_table WHERE product\_category = 'books', Snowflake may have to scan all of the micro-partitions to find all the relevant rows, which can be inefficient for a very large table.

**Scenario 2: Table with a Cluster Key**

**1. Create Table:**

SQL

CREATE TABLE my\_large\_table (

id INT,

product\_category VARCHAR,

order\_date DATE

)

CLUSTER BY (product\_category, order\_date);

* **What happens:** Snowflake creates the logical table and notes the specified cluster key (product\_category, order\_date). This tells Snowflake that this table is a **clustered table** and that it should be actively managed.

**2. Insert Data:**

Now, we insert the exact same data as before.

SQL

INSERT INTO my\_large\_table (id, product\_category, order\_date) VALUES

(1, 'electronics', '2024-03-01'),

(2, 'books', '2024-01-15'),

(3, 'electronics', '2024-02-20'),

(4, 'clothing', '2024-01-05'),

(5, 'books', '2024-03-25');

* **What happens:** The data is initially loaded into micro-partitions in the order it was received, just like in the first scenario. However, because a cluster key is defined, Snowflake's **automatic clustering service** starts working in the background. It will reorganize the micro-partitions to ensure that data with similar product\_category and order\_date values are physically co-located.
* **Result:** Over time, the data for 'books' will be grouped together, and within that group, it will be sorted by order\_date. The same will happen for 'electronics' and 'clothing'. This makes queries that filter on these columns much more efficient.

For example, when you run SELECT \* FROM my\_large\_table WHERE product\_category = 'books', Snowflake can use the metadata from its micro-partitions to quickly identify and scan only the micro-partitions that contain 'books' data, skipping all others. This process is called **pruning** and is the main benefit of using a cluster key

**“For high-volume tables, I use clustering keys to optimize query performance. By aligning physical storage with common filter patterns, I reduce scan time and improve cost efficiency—especially in analytics-heavy environments.”**