**Case Study: Retail Analytics on Snowflake Cloud Data Platform**

**Background**

RetailCo, a mid-sized retail chain, wants to modernize its analytics environment. The company has large amounts of transactional sales data, product catalogs, customer information, and external feeds like online reviews. Previously, RetailCo relied on an on-premises data warehouse, which was costly to scale and slow to respond to seasonal demand surges.

RetailCo decides to migrate to **Snowflake Cloud Data Platform** to gain scalability, elasticity, and simplified data management.

**1. Snowflake Cloud Data Platform Overview**

Snowflake is a fully managed cloud data warehouse running on AWS, Azure, or GCP. Its architecture separates **storage**, **compute**, and **services** layers:

* **Storage**: Centralized storage for structured/semi-structured data.
* **Compute (Virtual Warehouses)**: Independent clusters that scale up/down on demand.
* **Services**: Handles authentication, metadata, query optimization, and access control.

RetailCo chose Snowflake to:

* Handle **seasonal sales spikes** (scale compute on demand).
* Use **semi-structured formats** (e.g., JSON product reviews).
* Enable **time travel** for recovering data after accidental deletes.

**2. Virtual Warehouses: Compute Separation from Storage**

RetailCo sets up **two virtual warehouses**:

* **ETL\_WH**: For nightly data loading (medium-sized, auto-suspend after 15 minutes).
* **BI\_WH**: For analysts running dashboards (large warehouse, auto-scale enabled).

Key Benefits:

* ETL jobs don’t affect BI queries (separate compute).
* Costs are controlled with **auto-suspend/resume**.

Example command:

CREATE WAREHOUSE BI\_WH

WITH WAREHOUSE\_SIZE = 'LARGE'

AUTO\_SUSPEND = 300

AUTO\_RESUME = TRUE;

**3. Databases, Schemas, and Tables**

RetailCo structures its Snowflake environment as follows:

* **Database**: RETAIL\_DB
* **Schemas**:
  + SALES (transaction data)
  + CUSTOMER (profiles & preferences)
  + PRODUCT (catalog, reviews)

**Table Types**

1. **Standard Table** (persistent)
2. CREATE TABLE SALES.TRANSACTIONS (
3. SALE\_ID INT,
4. CUSTOMER\_ID INT,
5. PRODUCT\_ID INT,
6. AMOUNT DECIMAL(10,2),
7. SALE\_DATE DATE
8. );

Used for storing long-term sales data.

1. **Transient Table** (no Fail-safe, cheaper storage, shorter retention)
2. CREATE TRANSIENT TABLE SALES.STAGING\_TRANSACTIONS (
3. SALE\_ID INT,
4. CUSTOMER\_ID INT,
5. PRODUCT\_ID INT,
6. AMOUNT DECIMAL(10,2),
7. SALE\_DATE DATE
8. );

Used for **staging ETL loads** where recovery is less critical.

**4. File Formats: CSV, JSON, Parquet, ORC**

RetailCo receives data in multiple formats:

* **CSV**: Daily point-of-sale exports.
* **JSON**: Customer feedback from e-commerce.
* **Parquet**: Product catalog from suppliers.
* **ORC**: External financial feeds.

Example File Format Creation:

CREATE FILE FORMAT my\_csv\_format

TYPE = 'CSV'

FIELD\_OPTIONALLY\_ENCLOSED\_BY = '"'

SKIP\_HEADER = 1;

CREATE FILE FORMAT my\_json\_format TYPE = 'JSON';

CREATE FILE FORMAT my\_parquet\_format TYPE = 'PARQUET';

CREATE FILE FORMAT my\_orc\_format TYPE = 'ORC';

**5. Data Loading Methods: COPY INTO from Stage**

Data is staged in **Snowflake Internal Stage** (@%table) or **External Stage** (AWS S3).

Example: Load CSV sales data from S3.

CREATE STAGE sales\_stage

URL = 's3://retailco-data/sales/'

FILE\_FORMAT = my\_csv\_format;

COPY INTO SALES.TRANSACTIONS

FROM @sales\_stage

ON\_ERROR = 'CONTINUE';

This enables **parallel, high-speed bulk loading**.

**6. Querying Data in Snowflake with SQL**

Once data is loaded, analysts can run standard SQL queries.

Examples:

* Total sales per product:
* SELECT PRODUCT\_ID, SUM(AMOUNT) AS TOTAL\_SALES
* FROM SALES.TRANSACTIONS
* GROUP BY PRODUCT\_ID
* ORDER BY TOTAL\_SALES DESC;
* JSON query for customer reviews:
* SELECT
* review:value::string AS review\_text,
* review:rating::int AS rating
* FROM PRODUCT.REVIEWS,
* LATERAL FLATTEN(input => json\_column);

**7. Time Travel & Cloning Basics**

RetailCo benefits from **Time Travel** to restore data after accidental deletions.

* Accidentally deleted yesterday’s transactions?
* UNDROP TABLE SALES.TRANSACTIONS;
* Query data as of 2 days ago:
* SELECT \*
* FROM SALES.TRANSACTIONS
* AT (OFFSET => -2\*24\*60);
* Create a **zero-copy clone** for UAT testing:
* CREATE TABLE SALES.TRANSACTIONS\_CLONE CLONE SALES.TRANSACTIONS;

This avoids data duplication costs.

**Business Impact for RetailCo**

* **Performance**: Query times reduced by 60% due to independent scaling of BI\_WH.
* **Cost Optimization**: Staging data in transient tables + auto-suspend warehouses saved ~30%.
* **Agility**: Analysts can work on clones without affecting production.
* **Resilience**: Time travel enabled recovery from accidental table drops.

**Conclusion**

By adopting **Snowflake**, RetailCo gained:

* Elastic compute with **Virtual Warehouses**.
* Organized environment with **databases, schemas, and tables**.
* Support for multiple **file formats**.
* Simplified **data loading with COPY INTO**.
* Flexible **SQL querying** for structured and semi-structured data.
* Data resilience with **time travel and cloning**.

This modernized RetailCo’s analytics platform, enabling faster insights, cost control, and business scalability.