Welcome to your new dbt project!

Webshop Data Warehouse POC

1. Overview

This project is a **proof-of-concept data warehouse** built on Snowflake, orchestrated with dbt. It demonstrates an end-to-end pipeline from raw CSV landing through business-ready marts, supporting the CEO's key metrics:

- **Order Intake & Revenue** by shop, location, time
- **Drop-off Rate** from intake to shipped revenue
- **Shipping Lag** (purchase → delivery)
- **New vs. Returning Customers** over time

---

2. Architecture & Ingestion

2.1 PoC Ingestion (Raw Schema)

- **Schema**: `raw`
- **Method**: Snowflake **Dynamic Tables** + **ENABLE_SCHEMA_EVOLUTION**
- **Process**:
  1. Stage CSV files in an internal stage (`@raw/...`).
  2. Create dynamic tables on top of the stage; Snowflake infers schema and picks up new columns automatically.
  3. Changed or added files automatically insert or evolve columns on reload.

> **Why?**
> Quick to set up, no external orchestration required, perfect for a 3-hour PoC.

2.2 Production-Grade Ingestion (Future)

- **Schema**: `raw`
- **Method**: External S3 stage + Snowpipe + **schema evolution** + **PIPES**
- **Process**:
  1. Drop CSVs into an S3 bucket.
  2. Snowpipe auto-ingests new files into raw tables.
  3. Schema evolution handles new columns.
  4. Provides event-driven, near real-time loading.

---

3. dbt Layered Schema Design

```
raw   →  stg   →  core   →  mart

| Layer     | Schema | Purpose                                                    |
| -------- | ------ | ------------------------------------------------- |
| **Raw**  | `raw`  | Landing zone & schema-evolved ingest              |
| **Staging**  | `stg`  | Light cleaning, renaming, type casting (`stg_*`)
|
| **Core** | `core` | Star schema dims/facts (incremental: `d_*`, `f_*`) |
| **Mart** | `mart` | Business aggregations & KPI views (`rep_*`)       |
```

4. Naming Conventions

```
Staging models → schema stg, tables named stg_<source>

Core Dimensions → schema core, tables named d_<business_object>

Core Facts → schema core, tables named f_<fact_name>

Marts → schema mart, views named rep_<metric_name>

dbt models folders mirror schemas: models/stg/, models/core/, models/mart/
```

5. Key Findings & Assumptions

5.1 Products (`stg_products` → `core.d_products`)

- **`product_number`** appears to be the *base product* code.
- When **`is_variant = TRUE`**, each `sku_id`/`product_id` is a *variant* (e.g. color, size).
- When **`is_variant = FALSE`** but `product_number` still has multiple rows, these are likely **historical versions** of the base product (tracked by `updated_at`).
- **Assumption:** Keep only the **latest** row per `sku_id` and link every variant back to its base via `product_number`.
- **Surrogate key:** `product_sk` generated via `dbt_utils.surrogate_key`.

5.2 Customers (`stg_customers` → `core.d_customers`)

- Multiple `customer_id`s may represent the **same human** (guest vs. registered).
- No reliable email/name fields to dedupe; treat each `customer_id` as **distinct**.
- **Future:** Implement identity resolution via email/name fuzzy matching or an external MDM system.
- **Incremental logic:** Keep only the **latest** row per `customer_id` based on `updated_at`.
- **Surrogate key:** `customer_sk`.

5.3 Orders (`stg_orders` → `core.f_orders`)

- **`order_date`** = event for **Order Intake**
- **`delivery_date`** = event for **Revenue Recognition**
- **Shipping Lag** = `DATEDIFF('day', order_date, delivery_date)`
- Various `sales_event` statuses (e.g. `shipped`, `return`, `failed_payment`, etc.).
    - **Drop-off Rate** uses only `webshop_order` → `shipped` for intake vs. revenue.

    - Other statuses (returns, failed, etc.) are tracked separately or assumed out-of-scope for basic drop-off.

5.4 Order Positions (`stg_order_positions` → `core.f_order_lines`)

- **`price`** & **`quantity`** used to compute `position_amount = quantity * price`.
- **Precision:** Cast to **`DECIMAL(10,4)`** and `ROUND(price,4)` for all monetary/unit fields; final line amounts stored as **`DECIMAL(12,2)`**.
- **Unit of Measure:** Assumed price is **per-unit** in EUR.
- **Ambiguity:** If price were total per line, use `price / quantity`; we assume per-unit for the PoC.

5.5 Shops (`stg_shops` → `core.d_shops`)

- Static lookup of shop metadata: `shop_id`, `shop`, `platform`, `locale`, etc.
- **Materialization:** Full-refresh table; very small, so rebuild on every run.

5.6 Date Dimension (`core.d_date`)

- Covers **`1990-01-01`** → 70 years out.
- All timestamps loaded as **NTZ**; business dates assume **UTC**.
- **Surrogate key:** `date_sk = YYYYMMDD`.
- Supports both **`order_date`** and **`ship_date`** keys.

---

6. Marts & Example Metrics

- **`rep_order_intake_vs_revenue`**
  Intake, revenue, and drop-off rate by date/shop/location.

- **`rep_shipping_lag`**
  Average & median shipping lag by date/shop.

- **`rep_customer_retention`**
  Absolute & relative new vs. returning customers over time.

---

7. Next Steps & Production Considerations

- Automate ingestion with **Snowpipe** & external stages + pipes for true CDC.
- Improve customer dedupe via **MDM** or fuzzy-matching on email/name/address.
- Implement **SCD Type 2** for dimensions that require history.
- CI/CD & Testing: Add dbt tests (`not_null`, `unique`, `relationships`) and integrate with GitHub Actions.
- Enforce **RBAC** and data governance in Snowflake.
- Optimize performance: clustering keys on fact tables, right-size warehouses.