Weather driven data bot POC - Data Creation

Executive summary

We will combine the client's **WDD metrics fact** (`product × location × end_date` with `metric`, `metric_nrm`) and their dimensions (**product hierarchy** `phier`, **location attributes** `locdim`, **calendar** `cal`) with a compact set of datasets we own (Transactions, Events, Weekly Weather Conditions, Perishability, Inventory, Product Map). This delivers prompt-ready answers and actions: explain last week, anticipate next week, and recommend order/transfer/markdown moves.

Everything is designed for **minimal or zero manual intervention** as we move from POC to production.

Client inputs used:

- **Metrics** fact: `product, location, end_date, metric, metric_nrm`
- **Product hierarchy**: `dept, category, product` (`phier`)
- **Location attributes**: `region, market, state, location` (`locdim`)
- **Calendar**: `end_date` (week ending), plus `week/month/quarter/year` (`cal`)

Scope & assumptions

- **Granularity: Weekly**, aligned to retail **4-5-4**; we retain **week end_date** from `cal` as the canonical time key and derive `week_start_date` for convenience.
- **Geography:** U.S. markets/locations per `locdim`; `location` is the atomic geo key. `market` / `state` / `region` are **parallel** rollups (not strictly hierarchical).
- **Principles:** derived effects (no hard-coded "weather→SKU" lists), deterministic joins, append-only pipelines, automated refresh & DQ.
- **Core join keys:** `product`, `location`, `end_date` (plus `store_id`, `product_id` where applicable in our synthetic layers).

Data products we will create

1. **Synthetic Transactions** (weekly store×product)
2. **Events Calendar** (holidays, sports, civic)
3. **Weekly Weather Conditions** (actual weather summaries & anomaly flags)
4. **Perishability & Shelf-life** (product attributes)
5. **Inventory Layer** (Snapshot + Policy + Supply)
6. **Location Attributes (client `locdim`)** *(used as-is; documented here for completeness)*
7. **Product Mapping** (internal IDs ↔ product strings)

> For each dataset below: Purpose → Grain & keys → Schema → POC approach → Production automation.

1) Synthetic Transactions (weekly store×product)

Purpose

Provide realistic "actuals" to correlate with WDD, power variance explainers, and enable inventory decisions.

Grain & keys

- Grain: `store × product × week`
- Keys: `store_id`, `product_id`, `week_end_date` (carry `product`, `location` for joins)

Schema

Column	Type	Req	Notes / Example
week_start_date	DATE	✓	Derived from `end_date` (Sun).
week_end_date	DATE	✓	**Join to `cal.end_date`** (Sat).
location	STRING	✓	**Atomic geo key** from `locdim` (e.g., ST####).
market	STRING		From `locdim` (display/rollup).
store_id	STRING	✓	Stable ID (client or synthetic).
product_id	STRING	✓	Our internal ID.
product	STRING	✓	**Exact product string** (joins to `phier` /metrics).
units_sold	INT	✓	Weekly units; no negatives.
revenue_usd	NUM		`units×price` .
price_per_unit	NUM		Optional realism/elasticity.
promo_flag	BOOL		Sparse; avoids swamping WDD.
stockout_flag	BOOL		Sparse; lost-sales signal.

POC approach

- **Programmatic synthesis** only for `(product, location, end_date)` combos present in **metrics**:
  - **Baseline (LY)** per `product×market` seasonal curve + holiday pulses.
  - **This year**: `sales = baseline + weather_delta` (where `weather_delta = metric − metric_nrm`) `+ ε` .
  - **Store split**: allocate market totals to stores via fixed weights so `Σ stores = market` .

Production automation

- Swap synthetic with **client POS** (same schema & keys).
- **Weekly/daily** ingestion; idempotent upsert; reconciliation rules ensure market totals = Σ stores; correlation checks with WDD.

> Why changed earlier: replaced planalytics_product_label with product and added location to align with client fact/dims.

2) Events Calendar (holidays, sports, civic)

Purpose

Tag **non-weather** drivers; combine with weather for forward calls and better explanations.

Grain & keys

- Grain: Event record (exploded to `geo × week` for joins)
- Keys: `event_id` (joins via `(market OR location, week_end_date)`)

Schema

Column	Type	Req	Notes / Example
event_id	STRING	✓	Stable key.
event_name	STRING	✓	“Independence Day”, “Super Bowl”.
event_type	ENUM	✓	`holiday` `sport` `festival` `civic` .
start_date	DATE	✓	Local date.
end_date	DATE	✓	Inclusive.
geo_scope	ENUM	✓	`national` `state` `market` .
geo_value	STRING	✓	`USA` , `NY` , `Los Angeles` .
source_url	STRING	✓	Provenance/traceability.

POC approach

- **Scraped** authoritative sources (OPM ICS, timeanddate, official league sites, city portals). Normalize to canonical `market/state` ; explode to weeks.

Production automation

- **Monthly** jobs: ICS/API pulls; season schedule updates; idempotent upserts & dedupe; all rows carry `source_url`.

3) Weekly Weather Conditions (market×week)

Purpose

Explain *what weather happened* (heatwave, rain, snow); power condition-specific prompts; validate WDD spikes.

Grain & keys

- Grain: `market × week_end_date`
- Keys: `market`, `week_end_date`

Schema

Column	Type	Req	Notes / Example
week_end_date	DATE	✓	Join to `cal.end_date`.
market	STRING	✓	From `locdim`.
avg_temp_f	NUM		Weekly mean (°F).
temp_anom_f	NUM		Deviation vs normal (°F).
tmax_f	NUM		Weekly max (°F).
tmin_f	NUM		Weekly min (°F).
precip_in	NUM		Weekly precipitation (in).
precip_anom_in	NUM		Deviation vs normal (in).
heatwave_flag	BOOL		e.g., `temp_anom_f ≥ +10`.
cold_spell_flag	BOOL		e.g., `temp_anom_f ≤ −10`.
heavy_rain_flag	BOOL		Thresholded.
snow_flag	BOOL		True if measurable snow.
source_system	STRING	✓	“NOAA”, “OWM”, “VC”.

POC approach

- **API**: NOAA/NCEI preferred; aggregate daily → weekly; compute anomalies; set flags by thresholds.

Production automation

- **Weekly** pulls with retries/fallback/cache; partition by week; coverage/gap-fill (nearest station) and unit checks.

4) Perishability & Shelf-life (product attributes)

Purpose

Control stocking horizon, pull-forward behavior, and markdown urgency by product type.

Grain & keys

- Grain: product
- Keys: **`product`**

Schema

Column	Type	Req	Notes / Example
product	STRING	✓	**Exact product string** (joins to `phier`/metrics).
perishable_flag	BOOL	✓	True/False.
storage_band	ENUM	✓	`ambient` `chilled` `frozen`.
shelf_life_days	INT	✓	Typical unopened life.
recommended_cover_days	INT		Policy hint.
notes	STRING		Source/rationale.

POC approach

- **Reference** table compiled once from public guidance; assumptions documented.

Production automation

- Append when new **products** appear; optionally replace/augment from client product master; lightweight review workflow.

5) Inventory Layer (Snapshot + Policy + Supply)

Purpose

Turn insights into **orders, transfers, markdowns** with simple, explainable rules.

Grain & keys

- Snapshot grain: `store × product × week_end_date`
- Keys: `store_id`, `product_id`, `week_end_date`

5a) Inventory_Snapshot — Schema

Column	Type	Req	Notes / Example
week_end_date	DATE	✓	Joins to `cal.end_date`.
store_id	STRING	✓	Store identifier.
location	STRING		Optional back-reference to `locdim`.
product_id	STRING	✓	Product identifier.
product	STRING	✓	Exact product string (for joins).
on_hand_units	INT	✓	Units physically available.
in_transit_units	INT	✓	Units arriving ≤ horizon.
reserved_units	INT		Optional reservations.
exp_date	DATE		For perishables (if tracked).
shelf_capacity_units	INT		Optional capacity.

POC approach

- **Synthetic** seeding (cover by perishability band); roll forward `on_hand_next = on_hand + receipts − sales`.

Production automation

- **Nightly** ingest from ERP/WMS (on-hand, POs, transfers); incremental updates; SLA monitoring.

5b) Replenishment_Policy — Schema

Column	Type	Req	Notes / Example
product_id	STRING	✓	Policy at product or product×market/DC.
market_or_dc	STRING	✓	Scope for policy.
lead_time_days	INT	✓	e.g., 7.
review_cycle_days	INT	✓	e.g., 7.
min_cover_wks	NUM	✓	e.g., 1.0.
max_cover_wks	NUM	✓	e.g., 2.5.
case_pack	INT	✓	Rounding constraint.
moq	INT		Minimum order qty.
reorder_multiple	INT		e.g., 12.
service_level_target	NUM	✓	e.g., 0.95.

POC approach

- **Config table** with defaults by perishability band.

Production automation

- Managed as governed master data; edited via UI or controlled file; versioned.

5c) Supply_Schedule — Schema

Column	Type	Req	Notes / Example
po_id	STRING	✓	PO/transfer ID.
src	STRING	✓	DC/store source.
dest_store_id	STRING	✓	Destination store.
product_id	STRING	✓	Product.
product	STRING	✓	Exact product string (for joins).
eta_week_end	DATE	✓	Expected arrival week end_date.
qty	INT	✓	Units.

POC approach

- **Synthetic** POs created when order logic triggers; ETA = `now + lead_time_days`.

Production automation

- Ingest open POs/transfers from ERP/WMS; auto-reconcile receipts vs plans.

6) Location Attributes (client `locdim`)

Purpose

Standardize geography and enable clean roll-ups and joins.

Grain & keys

- Grain: location
- Keys: **`location`**

Schema

Column	Type	Req	Notes / Example
location	STRING	✓	**Atomic geo key**; joins to metrics.
market	STRING	✓	Display/rollup label (not strictly hierarchical with state).
state	STRING	✓	Two-letter code.
region	STRING		Client grouping.

POC approach

- **Client** `locdim` as source; use as-is.

Production automation

- **Monthly** sync from client master; auto-append new/closed locations; referential checks against Transactions & Inventory.

> Why changed earlier: replaced prior “store→city→region” with the client’s locdim shape and key location.

7) Product Mapping (IDs ↔ product strings)

Purpose

Deterministic joins across all tables and optional mapping to internal IDs.

Grain & keys

- Grain: product
- Keys: `product_id`, **`product`**

Schema

Column	Type	Req	Notes / Example
product_id	STRING	✓	Internal ID.
product	STRING	✓	**Exact product string** from metrics/phier.
category	STRING		From `phier`.

Column	Type	Req	Notes / Example
dept	STRING		From `phier`.
uom	STRING		“Units”, etc.

POC approach

- **Derived** from `phier`/metrics; assign IDs; mirror product strings exactly.

Production automation

- Auto-append when new products appear; optional SCD mapping to client catalog.

> Why changed earlier: category/dept sourced directly from phier; keys align to product.

How it all fits together (joins & flows)

- **Client star:** `metrics` fact joins to `phier` (on **product**), `locdim` (on **location**), `cal` (on **end_date**).
- **Our layers:**
  - **Transactions ↔ metrics:** attribute ΔvsLY to **weather** using `weather_delta = metric − metric_nrm`.
  - **Events & Weekly Weather ↔ Transactions/metrics:** narrative & driver decomposition by `(market/location, end_date)`; **no pre-tagged effects**.
  - **Perishability ↔ WDD forecast:** timing rules (pull-forward vs just-in-time; markdown urgency).
  - **Inventory ↔ Transactions/WDD:** order/transfer/markdown recommendations (lead times, pack sizes).
  - **Product Map/`phier`:** clean rollups to category/department.

Example prompt wiring

- **“What uplift last week for X in Market Y?”** → `metrics` (Σ vs normal) + Transactions; add Weekly Weather note (“+12°F vs normal”).
- **“Heatwave next week—what to stock?”** → `metrics` forecast + Weekly Weather flags + Perishability; pull-forward long-life, tighten perishable timing.
- **“Explain spike in Market Z.”** → Transactions + `metrics` + Events + Weekly Weather; attribute split.
- **“Do we have enough? What to order?”** → Inventory + Policy + `metrics` forecast; order rounded to case pack; suggest transfers.
- **“Markdown candidates?”** → negative WDD + short shelf-life + high on-hand.

Acceptance checks (each refresh)

- **Joinability:** 100% of **metrics** rows match a **product** in `phier` and a **location** in `locdim`; all rows map to a **calendar** **`end_date`** (no orphans).
- **Calendar integrity:** all facts map to valid weeks; TY↔LY comp table built; 52/53 rows per year.
- **Roll-ups:** market totals = Σ stores; region = Σ markets (≤0.1%).
- **Events/Weather coherence:** flagged weeks show plausible category movements.
- **Inventory math:** no negative on-hand; orders honor `case_pack`; ETA logic consistent.

Appendix A — Column reference

Transactions.csv

`week_start_date (DATE, req) • week_end_date (DATE, req) • location (STRING, req) • market (STRING) • store_id (STRING, req) • product_id (STRING, req) • product (STRING, req) • units_sold (INT, req) • revenue_usd (NUM) • price_per_unit (NUM) • promo_flag (BOOL) • stockout_flag (BOOL)`

Events.csv

`event_id (STRING, req) • event_name (STRING, req) • event_type (ENUM: holiday|sport|festival|civic, req) • start_date (DATE, req) • end_date (DATE, req) • geo_scope (ENUM: national|state|market, req) • geo_value (STRING, req) • source_url (STRING, req)`

WeatherConditions.csv

`week_end_date (DATE, req) • market (STRING, req) • avg_temp_f (NUM) • temp_anom_f (NUM) • tmax_f (NUM) • tmin_f (NUM) • precip_in (NUM) • precip_anom_in (NUM) • heatwave_flag (BOOL) • cold_spell_flag (BOOL) • heavy_rain_flag (BOOL) • snow_flag (BOOL) • source_system (STRING, req)`

Perishability.csv

`product (STRING, req) • perishable_flag (BOOL, req) • storage_band (ENUM: ambient|chilled|frozen, req) • shelf_life_days (INT, req) • recommended_cover_days (INT) • notes (STRING)`

Inventory_Snapshot.csv

`week_end_date (DATE, req) • store_id (STRING, req) • location (STRING) • product_id (STRING, req) • product (STRING, req) • on_hand_units (INT, req) • in_transit_units (INT, req) • reserved_units (INT) • exp_date (DATE) • shelf_capacity_units (INT)`

Replenishment_Policy.csv

`product_id (STRING, req) • market_or_dc (STRING, req) • lead_time_days (INT, req) • review_cycle_days (INT, req) • min_cover_wks (NUM, req) • max_cover_wks (NUM, req) • case_pack (INT, req) • moq (INT) • reorder_multiple (INT) • service_level_target (NUM, req)`

Supply_Schedule.csv

`po_id (STRING, req) • src (STRING, req) • dest_store_id (STRING, req) • product_id (STRING, req) • product (STRING, req) • eta_week_end (DATE, req) • qty (INT, req)`

LocDim.csv (client)

`location (STRING, req) • market (STRING, req) • state (STRING, req) • region (STRING)`

Phier.csv (client)

`dept (STRING, req) • category (STRING, req) • product (STRING, req)`

ProductMap.csv (optional)

`product_id (STRING, req) • product (STRING, req) • category (STRING) • dept (STRING) • uom (STRING)`

(Reference) Metrics.csv (client)

`product (STRING, req) • location (STRING, req) • end_date (DATE, req) • metric (NUM, req) • metric_nrm (NUM, req)`