**1) Post-load validation (in Azure SQL)**

**Goal:** prove the raw layer is correct before building anything on top.

What to check:

* **Row counts** match expectations (≈ 103k customers, 12k products, 300k orders, ~900k order\_items, ~152k returns).
* **PK uniqueness** (e.g., no duplicate order\_item\_id).
* **Referential integrity**
  + Every raw\_order\_items.order\_id exists in raw\_orders.order\_id.
  + Every raw\_returns.order\_item\_id exists in raw\_order\_items.order\_item\_id.
* **Domain checks**
  + quantity > 0, unit\_price ≥ 0, discount ≥ 0, tax ≥ 0.
  + shipping\_region ∈ {APAC, EMEA, AMER, LATAM}.
* **Date window sanity**
  + No return\_date before order\_date.
  + order\_date range looks 2022–2025 (remember: it’s NVARCHAR in raw).

👉 If you want a ready-to-paste **validation query pack**, say **“show validation queries.”**

**2) Create clean curated views (star schema)**

**Goal:** expose BI-ready tables while keeping transformations server-side (perfect for query folding).

Create these **views**:

**a) vw\_dim\_customer**

* Start from raw\_customers.
* **Trim/Clean** first\_name, last\_name; keep email as key for dedupe.
* **Dedupe rule:** for the same email, keep the **latest updated\_at** row.
* Columns: customer\_key (use customer\_id for now), email, first\_name, last\_name, region, join\_date.

**b) vw\_dim\_product**

* Join raw\_products to raw\_vendors to bring vendor\_name, country.
* **Standardize category** (fix typos like “electrnics/eletronics” → “Electronics”; blanks → “Unknown”).
* Columns: product\_id, product\_name, category\_clean, subcategory, vendor\_id, vendor\_name, vendor\_country.

**c) vw\_dim\_date (either in SQL or create in Power BI)**

* Calendar table with one row per day covering your order/return ranges.
* Columns: Date, Year, Month, MonthName, Quarter, YYYYMM etc.
* Mark as **Date table** in Power BI later.

**d) vw\_fact\_order\_items\_daily**

* Join raw\_order\_items → raw\_orders to get order\_date (parse it!) and customer\_id, shipping\_region.
* **Parse date:** use a robust conversion (we’ll give you TRY\_CONVERT) to get order\_day (DATETIME/DATE).
* **Derive amounts:**
  + gross\_amount = quantity \* unit\_price
  + discount\_amount = discount
  + tax\_amount = tax
  + net\_sales = gross\_amount - discount\_amount
  + (you can also expose net\_with\_tax = net\_sales + tax\_amount if needed)
* **Daily grain:** group by order\_day, product\_id, customer\_id (and keep order\_id if you want detail).
* Columns (example): order\_day, order\_id, product\_id, customer\_id, shipping\_region, quantity, gross\_amount, discount\_amount, tax\_amount, net\_sales.

**e) vw\_fact\_returns**

* From raw\_returns, **parse dates** to proper types (already DATETIME2 in raw for returns).
* Join back to raw\_order\_items (by order\_item\_id) to bring unit\_price, quantity if you need extra fields.
* Derive metrics:
  + days\_to\_return = DATEDIFF(DAY, order\_date, return\_date)
  + days\_to\_refund = DATEDIFF(DAY, return\_date, refund\_processed\_at)
* Columns: order\_item\_id, order\_id, product\_id, qty\_returned, refund\_amount, reason, return\_date, days\_to\_return, days\_to\_refund.

👉 When you want the exact SQL for these views, say **“show the view scripts.”**

**3) Add minimal indexes (on RAW tables)**

**Goal:** snappy joins and DQ performance.

* raw\_order\_items(order\_id) and raw\_order\_items(product\_id).
* (Optional) raw\_orders(order\_id) is already PK; consider index on customer\_id if queries use it often.

Say **“show index scripts”** if you want the statements.

**4) Connect Power BI Desktop to the views**

**Goal:** keep transformations server-side; confirm query folding.

* **Get Data → SQL Server** → select only the **views** you created (vw\_\*).
* In **Power Query**, verify **View Native Query** is enabled at the last step.
* Create a **Date** table (if not in SQL) and mark it as **Date**.
* Relationships:
  + vw\_dim\_date[Date] → vw\_fact\_order\_items\_daily[order\_day]
  + vw\_dim\_date[Date] → vw\_fact\_returns[return\_date]
  + vw\_dim\_product[product\_id] → facts’ product\_id
  + vw\_dim\_customer[customer\_key] → facts’ customer\_id

**5) Core DAX measures (in PBI)**

**Goal:** business KPIs.

* Total Sales (sum of net\_sales)
* Total Returns (sum of refund\_amount)
* Net Revenue After Returns (Total Sales – Total Returns)
* Refund % (DIVIDE(Total Returns, Total Sales))
* Units Returned % (returned qty vs sold qty)
* YoY growth, MTD, rolling 30/90 (once base measures are correct)

If you want the exact measure list later, say **“show DAX measures.”**

**6) Composite model + Incremental Refresh**

**Goal:** history fast (Import), latest fresh (DirectQuery).

* Add **RangeStart/RangeEnd** parameters in Power Query.
* Apply a date filter on vw\_fact\_order\_items\_daily.order\_day using those params (folding).
* Configure **Incremental Refresh**: e.g., **store 24 months, refresh 1 month**.
* Turn the fact into **Hybrid** (latest N days by DirectQuery).
* Validate: old months query instantly; latest days pull from Azure SQL.

Want a step list for this? Say **“show IR steps.”**

**7) Publish & set up the Service**

* Publish dataset/report to **Power BI Service**.
* Configure **SQL credentials** (no gateway needed for Azure SQL).
* Set **Scheduled Refresh**.
* (Optional) **RLS** by region/vendor; test with **View as Role**.
* Add a **KPI email subscription** for stakeholders.

**Tiny tracker (copy/paste)**

NEXT-STEPS TRACKER

[ ] Post-load validations passed (counts, PKs, FKs, domain, dates)

[ ] vw\_dim\_customer defined (dedupe by latest updated\_at)

[ ] vw\_dim\_product defined (category standardized, vendor joined)

[ ] vw\_dim\_date ready (SQL or PBI)

[ ] vw\_fact\_order\_items\_daily defined (parsed dates, amounts)

[ ] vw\_fact\_returns defined (days\_to\_return/refund)

[ ] Indexes created on raw\_order\_items(order\_id, product\_id)

[ ] Power BI connected to views; folding confirmed

[ ] Relationships built; Date marked as date table

[ ] Core DAX measures created and validated

[ ] Incremental Refresh + Hybrid configured

[ ] Published to Service; refresh scheduled; (optional) RLS set

Great “real-world” question. Here’s a tight playbook you can reuse when **duplicates + nulls** show up everywhere.

**0) Don’t panic—protect the raw layer**

* Keep **RAW** tables as an **immutable landing zone** (no hard PK/FK yet).
* Add ingestion metadata: batch\_id, load\_dttm, source\_file.
* Do all fixes in **curated views/tables** upstream.

**1) Profile & quantify (fast)**

* Row counts, null %, duplicate counts, FK orphans, min/max ranges.
* Prioritise **key columns** (\*\_id, dates, amounts, codes).

*Templates*

-- Null % (example)

SELECT

SUM(CASE WHEN customer\_id IS NULL THEN 1 ELSE 0 END)\*1.0/COUNT(\*) AS pct\_null\_customer\_id

FROM dbo.raw\_customers;

-- Duplicates by key (top offenders)

SELECT customer\_id, COUNT(\*) AS cnt

FROM dbo.raw\_customers

GROUP BY customer\_id

HAVING COUNT(\*) > 1

ORDER BY cnt DESC;

-- FK orphans

SELECT oi.order\_id

FROM dbo.raw\_order\_items oi

LEFT JOIN dbo.raw\_orders o ON o.order\_id = oi.order\_id

WHERE o.order\_id IS NULL;

**2) Define keys & survivorship rules (per entity)**

* **Business/Natural key** (e.g., email or customer\_external\_id) vs **Technical/Surrogate** (customer\_id).
* **Survivorship** for duplicates: keep **latest updated\_at**, or “most complete” row, or “trusted source wins”.
* Document this as your **data contract**.

**3) Build curated layer (views first, then optionally materialize)**

**a) De-duplicate views (ROW\_NUMBER)**

-- Keep latest per customer\_id

CREATE OR ALTER VIEW dbo.vw\_dim\_customer AS

WITH ranked AS (

SELECT \*,

ROW\_NUMBER() OVER (

PARTITION BY customer\_id

ORDER BY updated\_at DESC, join\_date DESC

) AS rn

FROM dbo.raw\_customers

)

SELECT customer\_id AS customer\_key,

LTRIM(RTRIM(first\_name)) AS first\_name,

LTRIM(RTRIM(last\_name)) AS last\_name,

email, region, join\_date, updated\_at

FROM ranked

WHERE rn = 1;

**b) Standardise dirty columns in views**

* Trim names, fix category spellings, default blanks to “Unknown”.
* Parse messy dates with TRY\_CONVERT/TRY\_CAST.

-- Example: parse raw\_orders.order\_date (NVARCHAR) to a clean date

CREATE OR ALTER VIEW dbo.vw\_orders\_clean AS

SELECT o.order\_id, o.customer\_id,

COALESCE(

TRY\_CONVERT(date, o.order\_date, 105), -- dd-mm-yy

TRY\_CONVERT(date, o.order\_date, 103), -- dd/mm/yy

TRY\_CONVERT(date, o.order\_date, 120), -- yyyy-mm-dd hh:mi:ss

TRY\_CONVERT(date, o.order\_date)

) AS order\_day,

o.shipping\_region

FROM dbo.raw\_orders o;

**4) Handle NULLs (by column criticality)**

* **Key/mandatory** (e.g., order\_id, product\_id): quarantine or map to **“Unknown”** dimension (**-1** key).
* **Semi-optional** (e.g., region): default ('Unknown') or infer from related data.
* **Measures** (amounts): treat missing as **0** only if the business agrees.

*Patterns*

-- Unknown dimension row (once)

-- INSERT INTO dim\_product(product\_id, product\_name, category, ...) VALUES (-1,'Unknown','Unknown',...);

-- Map orphans to -1 in a fact view

SELECT

CASE WHEN p.product\_id IS NULL THEN -1 ELSE oi.product\_id END AS product\_id\_key,

...

FROM dbo.raw\_order\_items oi

LEFT JOIN dbo.raw\_products p ON p.product\_id = oi.product\_id;

**5) Quarantine exceptions (never silently drop)**

* Create \_exceptions tables to **store bad rows + reason**.
* Use them in weekly triage with source owners.

-- Example: push duplicates to an exceptions table

SELECT rc.\*

INTO dbo.ex\_customers\_dupes

FROM dbo.raw\_customers rc

JOIN (

SELECT customer\_id

FROM dbo.raw\_customers

GROUP BY customer\_id

HAVING COUNT(\*) > 1

) d ON d.customer\_id = rc.customer\_id;

**6) Enforce constraints after cleaning**

1. Add **PKs/Unique indexes** on curated tables or materialized cleansed tables.
2. Add **FKs** (optionally start with **NOT FOR REPLICATION** or soft enforcement via QA).
3. Consider **filtered unique index** to enforce uniqueness only when a column is non-null.

-- After cleanliness proven:

ALTER TABLE dbo.dim\_customer

ADD CONSTRAINT PK\_dim\_customer PRIMARY KEY (customer\_key);

ALTER TABLE dbo.fact\_order\_items

ADD CONSTRAINT FK\_fact\_product FOREIGN KEY (product\_id\_key)

REFERENCES dbo.dim\_product(product\_id);

**7) Make loads idempotent (no double inserts)**

* Use **MERGE** or INSERT … WHERE NOT EXISTS with a stable natural key/hash.
* Keep a **load log** table to track batches.

-- Idempotent upsert pattern

MERGE dbo.dim\_customer AS t

USING (SELECT ... FROM dbo.vw\_dim\_customer) AS s

ON (t.customer\_key = s.customer\_key)

WHEN MATCHED THEN UPDATE SET ... -- survivorship fields

WHEN NOT MATCHED THEN INSERT (...) VALUES (...);

**8) Monitor & alert**

* Build a **Data Quality dashboard** (counts, null %, dupes, orphans, date outliers).
* Set thresholds & alerts (e.g., >0.5% orphan items triggers a warning).

**9) Iterate with source teams**

* Share exception samples & rules.
* Evolve the **data contract** (formats, mandatory fields, valid domain lists).

**Practical order for your project now**

1. Keep RAW as-is ✔
2. Create **vw\_orders\_clean** (parsed dates) and **vw\_dim\_customer** (dedup) ✔
3. Create **vw\_dim\_product** (standardised category + vendor join).
4. Create **vw\_fact\_order\_items\_daily** (join items→orders, derive amounts, map missing product\_id to -1 if any).
5. Optional: materialize views to tables, then add **PK/FK** there.
6. Connect Power BI to the **views/tables**, verify folding, then add DAX.

If you want, tell me **which entity** to start cleaning first (customers, orders, products, or facts) and I’ll give you a **ready-to-run script** for that one.

Yes—perfectly fine to park ingestion metadata for later. It’s nice-to-have for lineage/rollback, but not a blocker for analysis. Let’s keep moving.

## Next: Referential checks (orphans)

-- Items → Orders

SELECT COUNT(\*) AS orphans\_items\_orders

FROM dbo.raw\_order\_items oi

LEFT JOIN dbo.raw\_orders o ON o.order\_id = oi.order\_id

WHERE o.order\_id IS NULL;

-- Items → Products

SELECT COUNT(\*) AS orphans\_items\_products

FROM dbo.raw\_order\_items oi

LEFT JOIN dbo.raw\_products p ON p.product\_id = oi.product\_id

WHERE p.product\_id IS NULL;

-- Returns → Order Items

SELECT COUNT(\*) AS orphans\_returns\_items

FROM dbo.raw\_returns r

LEFT JOIN dbo.raw\_order\_items oi ON oi.order\_item\_id = r.order\_item\_id

WHERE oi.order\_item\_id IS NULL;

-- (Optional) sample the first few problem keys

SELECT TOP (50) oi.\*

FROM dbo.raw\_order\_items oi

LEFT JOIN dbo.raw\_orders o ON o.order\_id = oi.order\_id

WHERE o.order\_id IS NULL;

## Domain sanity

-- Quantities and money fields non-negative

SELECT

SUM(CASE WHEN quantity <= 0 THEN 1 ELSE 0 END) AS qty\_bad,

SUM(CASE WHEN unit\_price < 0 THEN 1 ELSE 0 END) AS price\_bad,

SUM(CASE WHEN discount < 0 THEN 1 ELSE 0 END) AS discount\_bad,

SUM(CASE WHEN tax < 0 THEN 1 ELSE 0 END) AS tax\_bad

FROM dbo.raw\_order\_items;

-- Regions only from the allowed set

SELECT shipping\_region, COUNT(\*) AS rows

FROM dbo.raw\_orders

GROUP BY shipping\_region

HAVING shipping\_region NOT IN ('APAC','EMEA','AMER','LATAM');

-- Optional: extreme discount flag (>90% off of list)

SELECT COUNT(\*) AS extreme\_discounts

FROM dbo.raw\_order\_items

WHERE discount / NULLIF(unit\_price\*quantity + discount,0) > 0.90;

## Date sanity

-- Returns must be on/after order\_date, and refunds on/after return\_date

SELECT

SUM(CASE WHEN return\_date < order\_date THEN 1 ELSE 0 END) AS bad\_return\_before\_order,

SUM(CASE WHEN refund\_processed\_at < return\_date THEN 1 ELSE 0 END) AS bad\_refund\_before\_return

FROM dbo.raw\_returns;

## Minimal indexes (for joins)

CREATE INDEX IX\_raw\_order\_items\_order\_id ON dbo.raw\_order\_items(order\_id);

CREATE INDEX IX\_raw\_order\_items\_product\_id ON dbo.raw\_order\_items(product\_id);

-- Optional if you often filter by customer:

-- CREATE INDEX IX\_raw\_orders\_customer\_id ON dbo.raw\_orders(customer\_id);

Run the three **checks** first and tell me if any counts are > 0. If they’re clean, we’ll create the curated **views** next.

 **batch\_id** → Tags every row with the *load run* it came from, so you can roll back, dedupe, and track lineage per batch.

 **load\_dttm** → Records the exact *UTC timestamp* of ingestion for auditing, incremental refresh, and SLA/debugging.

 **source\_file** → Captures the *origin file/path* so you can trace issues to a specific file and reproduce or quarantine it.

Its like doing EDA in sql

 **Referential checks (orphans)** → Make sure every child row has a valid parent (e.g., every order\_items.order\_id exists in orders). If not, joins break and metrics go wrong. Needed before adding FKs.

 **Domain sanity** → Catch impossible values (qty ≤ 0, negative prices/discount/tax, bad region codes). Stops garbage-in/garbage-out and protects your KPIs.

 **Date sanity** → Ensure timelines make sense (returns can’t be before orders; refunds can’t be before returns). Prevents negative durations and bogus time-series.

 **Minimal indexes** → Speed up joins/filters on big tables (order\_id, product\_id), which makes Power BI visuals/DirectQuery fast and keeps folding efficient.

If we find values in child tables here is the solution of what can we do

able.

**A) Fix upstream & reload (best)**

* Re-ingest missing parent rows (e.g., reload the Orders file for the missing IDs).
* Then re-run the orphan check (should drop to 0).

**B) Quarantine orphans (exclude from facts)**

-- Stash bad rows

SELECT oi.\*

INTO dbo.ex\_order\_items\_orphan\_orders

FROM dbo.raw\_order\_items oi

LEFT JOIN dbo.raw\_orders o ON o.order\_id = oi.order\_id

WHERE o.order\_id IS NULL;

-- In curated fact view, exclude them

-- ... WHERE o.order\_id IS NOT NULL AND p.product\_id IS NOT NULL

**C) Map to “Unknown” surrogate keys (keep totals intact)**

* Create **Unknown** rows in dimensions (key = **-1**), then map orphans to **-1** in the **view**.

-- Unknown rows once (only if you are creating physical dims later)

-- INSERT INTO dbo.dim\_product(product\_id, product\_name, category, subcategory, vendor\_id)

-- VALUES (-1, 'Unknown', 'Unknown', 'Unknown', -1);

-- In curated fact view (example skeleton):

SELECT

COALESCE(o.order\_id, -1) AS order\_id\_key,

COALESCE(p.product\_id, -1) AS product\_id\_key,

oi.order\_item\_id, oi.quantity, oi.unit\_price, oi.discount, oi.tax

FROM dbo.raw\_order\_items oi

LEFT JOIN dbo.raw\_orders o ON o.order\_id = oi.order\_id

LEFT JOIN dbo.raw\_products p ON p.product\_id = oi.product\_id;

* Keep a **metric** that tracks how many rows hit “Unknown” so you can monitor quality.

**D) Drop orphans (only with business sign-off)**

-- Dangerous if done on RAW; prefer to filter in views.

DELETE oi

FROM dbo.raw\_order\_items oi

LEFT JOIN dbo.raw\_orders o ON o.order\_id = oi.order\_id

WHERE o.order\_id IS NULL;

**3) Add trusted constraints after clean-up**

* When orphan counts == 0 (or you’re mapping to -1 in curated tables), then add FKs.
* Avoid WITH NOCHECK (it creates **untrusted** constraints). Prefer clean data + trusted FKs.

ALTER TABLE dbo.raw\_order\_items WITH CHECK

ADD CONSTRAINT FK\_raw\_items\_orders

FOREIGN KEY (order\_id) REFERENCES dbo.raw\_orders(order\_id);

ALTER TABLE dbo.raw\_order\_items WITH CHECK

ADD CONSTRAINT FK\_raw\_items\_products

FOREIGN KEY (product\_id) REFERENCES dbo.raw\_products(product\_id);

ALTER TABLE dbo.raw\_returns WITH CHECK

ADD CONSTRAINT FK\_raw\_returns\_items

FOREIGN KEY (order\_item\_id) REFERENCES dbo.raw\_order\_items(order\_item\_id);

**4) If the issue is formatting (fast fixes in curated layer)**

* **Trim/cast** keys in a view before joining:

-- Example for text feeds:

SELECT

TRY\_CAST(LTRIM(RTRIM(order\_id\_txt)) AS INT) AS order\_id\_clean,

...

FROM dbo.raw\_orders\_text;

* Then join order\_id\_clean to child tables.

**Practical recommendation for your project**

* Start with **B** (quarantine) or **C** (Unknown key) in the **views**, so analysis can proceed.
* In parallel, check if any parents are simply **missing due to import**—if yes, **reload** (A).
* Once the orphans drop to **0** (or are mapped consistently), add the **FKs**.

**Making clean views now**

**/\* 1) Customers — clean + dedupe \*/**

**CREATE OR ALTER VIEW dbo.vw\_dim\_customer AS**

**WITH ranked AS (**

**SELECT**

**customer\_id,**

**LTRIM(RTRIM(first\_name)) AS first\_name,**

**LTRIM(RTRIM(last\_name)) AS last\_name,**

**LOWER(LTRIM(RTRIM(email))) AS email,**

**LTRIM(RTRIM(region)) AS region,**

**join\_date,**

**updated\_at,**

**ROW\_NUMBER() OVER (**

**PARTITION BY customer\_id**

**ORDER BY updated\_at DESC, join\_date DESC, customer\_id**

**) AS rn**

**FROM dbo.raw\_customers**

**)**

**SELECT**

**customer\_id AS customer\_key,**

**first\_name,**

**last\_name,**

**email,**

**region,**

**join\_date,**

**updated\_at,**

**CONCAT(first\_name,' ',last\_name) AS full\_name**

**FROM ranked**

**WHERE rn = 1;**

**GO**

**/\* 2) Products — clean text + join vendors \*/**

**CREATE OR ALTER VIEW dbo.vw\_dim\_product AS**

**SELECT**

**p.product\_id,**

**LTRIM(RTRIM(p.product\_name)) AS product\_name,**

**CASE**

**WHEN LTRIM(RTRIM(p.category)) IS NULL OR LTRIM(RTRIM(p.category)) = '' THEN 'Unknown'**

**ELSE UPPER(LEFT(LTRIM(RTRIM(p.category)),1))**

**+ LOWER(SUBSTRING(LTRIM(RTRIM(p.category)),2,LEN(LTRIM(RTRIM(p.category)))))**

**END AS category,**

**LTRIM(RTRIM(p.subcategory)) AS subcategory,**

**p.vendor\_id,**

**v.vendor\_name,**

**v.country AS vendor\_country**

**FROM dbo.raw\_products p**

**LEFT JOIN dbo.raw\_vendors v**

**ON v.vendor\_id = p.vendor\_id;**

**GO**

**/\* 3) Order items — normalized amounts + clean order date \*/**

**CREATE OR ALTER VIEW dbo.vw\_fact\_order\_items\_clean AS**

**SELECT**

**oi.order\_item\_id,**

**oi.order\_id,**

**oc.order\_day, -- parsed date from vw\_orders\_clean**

**oc.customer\_id,**

**oc.shipping\_region,**

**oi.product\_id,**

**oi.quantity,**

**oi.unit\_price,**

**oi.discount,**

**oi.tax,**

**-- normalized amounts**

**ABS(oi.discount) AS discount\_amount,**

**CASE WHEN oi.discount < 0**

**THEN (oi.unit\_price \* oi.quantity) + oi.discount -- discount stored negative (off the list)**

**ELSE (oi.unit\_price \* oi.quantity) -- unit\_price already net of discount**

**END AS net\_sales,**

**CASE WHEN oi.discount < 0**

**THEN (oi.unit\_price \* oi.quantity) -- list was unit\_price\*qty**

**ELSE (oi.unit\_price \* oi.quantity) + oi.discount**

**END AS list\_amount,**

**CASE WHEN oi.discount < 0**

**THEN (oi.unit\_price \* oi.quantity) + oi.discount + oi.tax**

**ELSE (oi.unit\_price \* oi.quantity) + oi.tax**

**END AS net\_with\_tax**

**FROM dbo.raw\_order\_items oi**

**JOIN dbo.vw\_orders\_clean oc**

**ON oc.order\_id = oi.order\_id**

**WHERE oc.order\_day IS NOT NULL; -- keep only parsable dates**

**GO**

**/\* 4) Returns — durations & clean fields \*/**

**CREATE OR ALTER VIEW dbo.vw\_fact\_returns\_clean AS**

**SELECT**

**r.order\_item\_id,**

**r.order\_id,**

**r.product\_id,**

**r.qty AS qty\_returned,**

**r.refund\_amount,**

**LTRIM(RTRIM(r.reason)) AS reason,**

**CAST(r.return\_date AS date) AS return\_day,**

**DATEDIFF(DAY, CAST(r.order\_date AS date), CAST(r.return\_date AS date)) AS days\_to\_return,**

**DATEDIFF(DAY, CAST(r.return\_date AS date), CAST(r.refund\_processed\_at AS date)) AS days\_to\_refund**

**FROM dbo.raw\_returns r;**

**GO**

**/\* 5) Minimal date dim — from actual data points (good enough to relate) \*/**

**CREATE OR ALTER VIEW dbo.vw\_dim\_date\_from\_data AS**

**WITH d AS (**

**SELECT DISTINCT oc.order\_day AS [Date]**

**FROM dbo.vw\_orders\_clean oc**

**WHERE oc.order\_day IS NOT NULL**

**UNION**

**SELECT DISTINCT CAST(r.return\_date AS date)**

**FROM dbo.raw\_returns r**

**WHERE r.return\_date IS NOT NULL**

**)**

**SELECT**

**[Date],**

**DATEPART(YEAR, [Date]) AS [Year],**

**DATEPART(QUARTER,[Date]) AS [QuarterNum],**

**CONCAT('Q', DATEPART(QUARTER,[Date])) AS [Quarter],**

**DATEPART(MONTH, [Date]) AS [Month],**

**DATENAME(MONTH, [Date]) AS [MonthName],**

**DATEPART(DAY, [Date]) AS [Day],**

**(DATEPART(YEAR,[Date]) \* 100 + DATEPART(MONTH,[Date])) AS [YearMonth]**

**FROM d;**

**GO**

## What we already did to shape a ****star schema****

* **Grain defined:**
  + Fact = **order item** at **day** grain (order\_day), plus a Returns fact (return\_day).
* **Dimensions built:**
  + **Customer**: vw\_dim\_customer (trim, dedupe → one row per customer).
  + **Product**: vw\_dim\_product(s) (clean category, add vendor\_name/country).
  + **Date**: view or PBI Date table (marked as Date).
  + (Optional) **Vendor** as a separate dim if you prefer snowflake.
* **Clean facts:**
  + vw\_fact\_order\_items\_clean: join to clean orders date, normalize discounts (handles negative vs positive), derive net\_sales, list\_amount, net\_with\_tax.
  + vw\_returns\_clean: standardize reason, compute days\_to\_return/days\_to\_refund.
* **Keys & relationships (for PBI):**
  + Date[Date] → Facts’ [order\_day]/[return\_day] (one→many).
  + Product[product\_id] → Facts.
  + Customer[customer\_key] → Facts.
* **Performance helpers:**
  + Indexes on FK columns (raw\_order\_items(order\_id), (product\_id)), plus parsing date in a view to keep folding.
* **Data quality baked in:**
  + Trim/case fixes, Unknown buckets, date parsing, discount normalization, duplicate-handling in Customer.

If you want to **materialize one piece now** (e.g., a daily aggregate fact for ultra-fast dashboards), tell me which view and I’ll give a ready-to-run script + refresh pattern.

awesome—views are in. Here are **two clean ways to proceed**:

**1) Business-first path (pick a problem → KPIs → dashboard → actions)**

**A) Returns reduction (create the business problem)**

**Goal:** cut return rate by **20% in 90 days** without hurting sales.  
**Primary KPIs**

* **Return Rate (Qty):** qty\_returned / qty\_sold
* **Refund % (Value):** refund\_amount / net\_sales
* **Days to Return / Days to Refund**
* **Top Return Reasons** (share of refunds)
* **Hotspots:** by **vendor, category, product, region, month**

**Questions to answer**

* Which **products/vendors** drive 80% of refunds?
* Are **discount-heavy** items getting returned more?
* Which **reasons** dominate (size issue, defective, wrong item)?
* Where are **slow refunds** (customer pain)?

**Deliverables**

* **Returns Intelligence** dashboard: trend, heatmaps (vendor×category), reason Pareto, product drill-through.
* **Watchlist** of SKUs/vendors with: high return rate, large refund ₹, long days-to-refund.

**Actions**

* Tighten quality checks with top-3 vendors, adjust sizing charts, cap discounts for high-return SKUs, set SLA on refunds.

**B) Growth & pricing effectiveness**

**Goal:** grow **Net Sales +8% QoQ** with healthy discount discipline.  
**Primary KPIs**

* **Net Sales**, **List vs Net delta**, **Discount %**, **Gross Margin proxy** (list − net)
* **Price elasticity proxy:** sales vs discount bands
* **Product velocity** (units/day), **Attach rate** (if you add bundles later)

**Questions**

* Where do discounts **drive** volume vs where they just **burn** margin?
* Which categories/vendors respond best to small discounts?
* What’s the **optimal band** (e.g., 10–20%) per category?

**Deliverables**

* **Pricing & Discounts** dashboard: waterfall (List→Net→Refund), discount band analysis, top/bottom SKUs by “₹ per 1% discount”.

**Actions**

* Recalibrate discount rules per category/vendor; target limited promos.

**2) Tech-first path (step-by-step build & deploy)**

**Model & data (Power BI)**

1. **Load** only your vw\_\* views.
2. Create **Date** table (DAX) and **mark as Date**; relate to order\_day & return\_day.
3. **Relationships:**
   * Date → vw\_fact\_order\_items\_clean[order\_day]
   * Date → vw\_fact\_returns\_clean[return\_day]
   * vw\_dim\_product[product\_id] → facts
   * vw\_dim\_customer[customer\_key] → facts

**Core measures (names only; ask for code when ready)**

* Total Sales (SUM net\_sales)
* Total Refunds (SUM refund\_amount)
* Net After Returns = Sales − Refunds
* Units Sold, Units Returned
* Return Rate % (Qty) = Units Returned / Units Sold
* Refund % (Value) = Refunds / Sales
* Time intel: **YoY, MoM, MTD, QTD, YTD**, **Rolling 30/90**
* **Discount %** = Discount / List Amount
* **Avg Days to Refund**

**Report pages (suggested)**

* **Executive Overview:** Net Sales, Refunds, Return %, YoY; trend by month; slicers (Region, Category, Vendor).
* **Returns Intelligence:** reasons Pareto, vendor×category heatmap, long-tail product table (drill-through).
* **Pricing & Discounts:** list vs net waterfall, discount band analysis, “₹ per 1% discount”.
* **Product Performance:** velocity, seasonality by category, top/worst SKU cards.
* **Customer Lens (optional):** RFM summary (once you add Python features or simple SQL aggregates).

**Validation & QA**

* Cross-check PBI totals vs **SQL sanity queries** (same filters).
* Spot-check top 10 SKUs/vendors in both tools.

**Performance**

* Turn on **Incremental Refresh** (store 24 months, refresh 1 month).
* Consider **Hybrid tables** for last N days.
* Verify **query folding** at the last PQ step.
* Keep SQL indexes on order\_id, product\_id, and (if materialized) **date** columns.

**Deploy**

* Publish dataset/report, set **SQL credentials**, schedule refresh.
* (Optional) **RLS** by region/vendor; test “View as”.

**NOW connecting to powerbi**

**A) One-time prep (Azure)**

1. In Azure Portal → **SQL server** (not the DB) → **Networking**
   * Public access: **Allowed** (for now)
   * Add your **client IP** → **Save**
2. Note your connection info:
   * **Server**: <yourserver>.database.windows.net
   * **Database**: <yourdbname>
   * Auth: Azure AD (preferred) or SQL login

**B) Connect in Power BI Desktop (DirectQuery)**

1. Home → **Get Data** → **Azure** → **Azure SQL Database**
2. Enter **Server** and **Database**
3. **Data Connectivity mode**: choose **DirectQuery** → **OK**
4. Sign in (Azure AD/Organizational or Basic for SQL login)
5. In Navigator, select **only your views**:
   * vw\_fact\_order\_items\_clean
   * vw\_fact\_returns\_clean
   * vw\_dim\_products (or vw\_dim\_product)
   * vw\_dim\_customer
   * (either your DAX Date table or vw\_dim\_date\_from\_data)
6. Load.

**Build relationships**

* Date[Date] → Facts’ [order\_day] / [return\_day] (single direction)
* Product[product\_id] → Facts
* Customer[customer\_key] → Facts

**C) Composite model (DirectQuery fact + Import dims)**

If you want snappier slicers:

1. Connect as **DirectQuery** (as above).
2. Model view → select **dimension** tables → Storage mode **Dual** (or **Import**).
3. Keep **fact** tables **DirectQuery**.  
   This keeps filters fast while fact queries hit SQL live.

**D) Service setup (after Publish)**

* In Power BI Service → **Datasets** → your dataset → **Settings** → **Data source credentials**
  + For Azure SQL DB: pick **OAuth2 (Organizational)** or **Basic** (SQL login).
  + **Gateway not required** for Azure SQL DB with public endpoint.
* DirectQuery has no scheduled refresh (it’s live), but dashboard tiles use a cache—enable **Dashboard tile refresh** if needed.
* If you added any **Import** tables (composite), set a **scheduled refresh** for those.

**E) Performance tips for DirectQuery**

* Use **only the vw\_\* views** (we already cleaned/normalized them).
* Keep visuals light (fewer fields/measures per visual; fewer visuals per page).
* Prefer measures over calculated columns; push heavy calcs into SQL views.
* File → Options → **DirectQuery**: consider raising timeouts if your DB is small-SKU/large-range.
* Indexes you added (raw\_order\_items(order\_id), (product\_id)) already help; if you later **materialize** facts, add a **clustered columnstore index**

## How DirectQuery works

* Every visual triggers a **live SQL query** against your Azure SQL DB. No data is stored in the PBIX (only metadata/measures).
* Power Query steps are turned into SQL **if they fold**; otherwise Power BI pulls more data and processes locally (slow).

## Do Power Query changes reflect in Azure Data Studio?

* **No.** Power Query **never writes back** to the database. It only shapes the query **result** sent to Power BI.
* If you rename columns, filter rows, or compute a column in Power Query, you’ll **see it in Power BI only**. Azure Data Studio still shows the raw tables.

## Do DB changes reflect in Power BI (DirectQuery)?

* **Yes.** Update data in SQL → visuals re-query and show the latest (after you interact/refresh the page).
* In the Service, dashboard tiles may cache briefly; reports re-query on interaction.

## Best practice for your setup

* Put all heavy shaping in **SQL views** (vw\_\*) so it **folds 100%**.
* In Power Query, do **minimal** steps (rename, data types).
* Verify folding: Power Query step → **View native query** (if enabled).
* To see actual queries: use **Performance Analyzer** in PBI Desktop or run a trace in Azure Data Studio.

## TL;DR

* DirectQuery = live reads from SQL.
* Power Query changes **do not** modify the database.
* If you want the transformed data to exist in SQL/ADS, implement it as a **view or ETL** (not in Power

Great question. With **DirectQuery to Azure SQL**, the safest way to make client-requested changes is to treat your SQL views as the contract and promote changes through **dev → test → prod**. Here’s a practical playbook.

# 1) Pick the right path based on change type

**A) Visual-only (layout, slicers, bookmarks)**

* Open the report (thin report ideally), tweak visuals, no schema changes.
* Validate with **Performance Analyzer**.
* Publish to **Dev workspace** → client UAT → promote via **Deployment Pipeline**.

**B) Measure/logic changes (KPIs, DAX)**

* Edit measures in Desktop (or Tabular Editor).
* Unit-check a few numbers against SQL.
* Dev → UAT → Pipeline promote.

**C) Data/model changes (new columns, filters, joins)**

* **Change SQL views first** (CREATE OR ALTER VIEW). Keep names stable; **add** new columns rather than renaming/removing (backward compatible).
* In Desktop: **Transform data → Refresh preview → Close & Apply** (updates schema).
* Update visuals/measures to use the new fields.
* Dev → UAT → Pipeline promote. Remove old columns later in a second release.

# 2) DirectQuery best practices while changing

* **Do the heavy logic in SQL views**, keep Power Query minimal to preserve **folding**.
* Avoid breaking fields used by visuals (don’t rename/drop—add/alias instead).
* Use **parameters** for Server/DB so Dev/Test/Prod swap cleanly.
* Keep **dimensions Import/Dual**, facts DirectQuery (Composite) for snappy slicers.
* Test query load: DAX Studio (Server Timings) + ADS/Profiler.

# 3) Zero-downtime workflow (step-by-step)

1. **Branch** PBIX (or PBIP project) → Dev workspace copy.
2. If needed, **ALTER VIEW** in the **Dev DB**; preserve existing columns; add new ones.
3. Desktop: Apply changes, update visuals/measures; validate numbers vs SQL.
4. **UAT**: share the Dev report link with the client.
5. **Promote** to Test → Prod via **Deployment Pipelines** (keeps lineage/permissions).
6. **Rollback plan**: keep prior dataset/report version + SQL view script to revert.

# 4) Example mini-scenarios

* **“Add Gross Margin to the report”**  
  – Add gross\_margin to vw\_fact\_order\_items\_clean.  
  – Refresh schema in Desktop, create measure(s), update visual → Dev → UAT → promote.
* **“Rename Category to Product Category”**  
  – In SQL, **add** category AS product\_category (alias) and keep category for now.  
  – Update visuals to the new field.  
  – In a later release, drop the old one.

# 5) Tooling most teams use

* **Power BI Deployment Pipelines** (Dev/Test/Prod).
* **Tabular Editor** (measures, calc groups, dynamic format strings).
* **ALM Toolkit** (dataset diffs).
* **Git** with **PBIP** projects for versioning.

**If you page is slow after making reports do the following steps**Here’s a battle-tested playbook, ordered from **fast triage → deeper fixes**. Do the top items first; they usually give 80% of the win.

# 0) Pinpoint what’s slow (5–10 min)

* **Performance Analyzer** (View ▸ Performance Analyzer): find the 2–3 slow visuals.
* **DAX Studio** (Server Timings + Query Plan): confirm if it’s **DAX** or **DirectQuery** (long SQL waits).
* **Power Query**: check **View Native Query** on each table (folding must be ON).

# 1) Quick report wins (no back-end change)

* **Reduce visuals per page** (aim ≤ 8) and fields per visual.
* Default the report to **Last 12 months** (Relative Date slicer) instead of “All time”.
* Use **Top N** + “Others” buckets on big tables.
* Turn off **Show items with no data** where not needed.
* Avoid high-cardinality slicers (Customer, Email). Use **search** or **filter pane** instead.
* Disable unnecessary **visual interactions** (Format ▸ Edit interactions).
* Options ▸ **Query reduction**: enable “Add Apply button to filters/slicers”.

# 2) Model tweaks (fast, safe)

* Keep a **star schema**; avoid bi-directional relationships (use **single** direction).
* **Dimensions = Import/Dual**, **Facts = DirectQuery** (Composite model) for fast slicers.
* Remove/hide unused columns; set correct **data types/encodings** (Fixed decimal for money).
* Turn off **Auto Date/Time** (Options ▸ Data Load).

# 3) DAX cleanups

* Prefer **native aggregations**: SUM( [net\_sales] ) over iterators (SUMX over large tables).
* Cache subexpressions with **VAR**; minimize nested CALCULATE.
* Be cautious with **DISTINCTCOUNT**; if needed, pre-aggregate in SQL.
* Avoid FORMAT() for numbers (returns text → slower); use **dynamic format strings** instead.

# 4) DirectQuery specifics

* Keep **all heavy logic in SQL views** (you already have vw\_\*): joins, derived columns, discount logic.
* Ensure **folding** end-to-end (if “View Native Query” is greyed out at the last step, move that step to SQL).
* Limit visuals that trigger **large result sets** (tables with thousands of rows). Paginate or drill-through.

# 5) Use a ****Composite + Aggregations**** design (big gains)

* Create a tiny **imported** table of **daily aggregates** (e.g., vw\_sales\_daily\_agg by order\_day, category, vendor).
* Map **Aggregations** so PBI uses Import for rollups and falls back to DQ for drill-through.
* Keep detail fact **DirectQuery** for rows/transactions.

# 6) Database-side boosts (Azure SQL)

* Index the columns used to **join/filter**:
* CREATE INDEX IX\_items\_order ON dbo.raw\_order\_items(order\_id);
* CREATE INDEX IX\_items\_product ON dbo.raw\_order\_items(product\_id);
* If you materialize a clean fact table, add **Clustered Columnstore Index** for scans:
* CREATE CLUSTERED COLUMNSTORE INDEX CCI\_fact\_order\_items ON dbo.fact\_order\_items;
* Add an **index (or partition)** on the **date** column used most in filters.
* Update statistics; check **Query Store** for expensive queries; consider rewriting views if needed.

# 7) When to switch mode

* If a page is still slow after the above, **Import** just that page’s dataset (or an **Import mini-mart view**) and keep the rest DQ (Composite model).
* Schedule refresh (e.g., hourly/daily). Client gets a snappy page plus live drill-through elsewhere.

# 8) Validate & lock in

* Re-run Performance Analyzer and compare timings before/after.
* Save slow/fast timings in your change log. If you use Deployment Pipelines, promote once green.

## Tiny example: applying this to your report

1. Default all pages to **Last 12 months**; add a “All time” toggle if needed.
2. Import **vw\_sales\_daily\_agg** and base overview cards/charts on it (instant).
3. Keep **detail table** DirectQuery for drill-through by order/customer.
4. Ensure all tables come from your **vw\_\*** (folding), and remove heavy M steps.
5. Add/confirm the two **FK indexes** above; if you later snapshot a fact, add **columnstore**.

If you want, tell me which page/visual is slow and what it shows—I’ll suggest the exact agg view (SQL), the fields to include, and how to wire Aggregations in Power BI so it speeds up immediately.

**----------------------------------------------------------------------**Current file ▸ Data Load: uncheck Auto Date/Time. why we need to do it?

###### ChatGPT said:

Unchecking **Auto Date/Time** is a best-practice because the built-in feature quietly creates a **hidden date table for every date column**. That causes problems:

* **Bloat & slowness:** dozens of hidden date tables = extra joins and a heavier model.
* **Inconsistent time logic:** each column gets its own tiny calendar (no single source of truth), so YoY/MTD can disagree across tables.
* **No fiscal/custom calendars:** auto tables are fixed to standard Gregorian; you can’t add holidays, fiscal years, etc.
* **DirectQuery pain:** those hidden date tables are **in-memory**; mixing them with DQ facts can hurt performance and folding.

**Do instead:** create **one proper Date table** (via DAX CALENDAR/CALENDARAUTO), add Year/Month/Fiscal columns, **Mark as date table**, and relate it to your facts. This gives consistent, faster time-intelligence everywhere.

**Also made two parameters**

**-servername**

**-dbname**

**POWER Query**

give me what all steps are reuqired in power query i haven tdone the mcode you gave me

###### ChatGPT said:

Thought for 17s

Awesome—here’s exactly what to do **inside Power Query** (click-by-click), no M code required beyond a tiny optional tweak.

# One-time setup (2 mins)

1. **Disable Auto Date/Time**: File → Options & settings → Options → Current file → **Data Load** → uncheck **Auto Date/Time**.
2. **Turn on folding indicators**: same Options window → **Power Query** → check **Show query folding indicators**.

# Create environment parameters (Dev/Test/Prod friendly)

1. Power Query Editor → **Manage Parameters** → **New**
   * **ServerName** (Text) → put your Azure SQL server (e.g., yourserver.database.windows.net)
2. **New** again → **DatabaseName** (Text) → yourdbname.

These just store the values so you don’t hardcode them in each query.

# Connect your tables/views (DirectQuery)

1. Home → **New Source** → **Azure** → **Azure SQL Database**.
2. Enter your **Server** and **Database**. Select **DirectQuery** → **OK**.
3. In Navigator, tick only the **views** you built:
   * vw\_fact\_order\_items\_clean
   * vw\_returns\_clean
   * vw\_dim\_products (or vw\_dim\_product)
   * vw\_dim\_customer
   * (Skip vw\_orders\_clean unless you really need it.)
4. Click **Transform Data** (not Load).

# For each query (repeat these steps)

1. **Rename** the query for clarity
   * e.g., Fact – Order Items, Fact – Returns, Dim – Products, Dim – Customer.
2. **Choose Columns** (Home → **Choose Columns**)
   * Keep only what you’ll use. (This folds to SELECT col1, col2….)
3. **Set Data Types** (Transform → **Data type**)
   * Dates → **Date** (order\_day, return\_day)
   * Money → **Fixed decimal number** (net\_sales, refund\_amount, etc.)
   * Keys → **Whole number** (order\_id, product\_id, customer\_key)
   * Text for names/categories/regions.
4. **Verify Folding**
   * Right-click the **last step** → **View Native Query**.
     + If **enabled** → ✅ good.
     + If **greyed out** → remove or move that step into the SQL view.
5. (Optional) **Don’t load Orders header**
   * If you brought in vw\_orders\_clean but won’t use it now: right-click it → **Enable load** (untick).

# (Optional) Swap hardcoded server/db to parameters (1-minute tweak)

If your Source step shows hardcoded values:

1. Select the query → **Advanced Editor**.
2. Replace Sql.Database("yourserver","yourdb") with Sql.Database(ServerName, DatabaseName).
3. Done. (Repeat for each query once.)

# Order of steps (keep it simple)

* **Source** → **Navigation** → **Choose Columns** → **Changed Type**.  
  Avoid extra steps (index columns, custom functions, merges to files) in DirectQuery—they often break folding.

# Close & Apply

Click **Close & Apply** to push changes back to the model.

## Quick sanity checks after Apply

* Open a table visual and drag order\_day + **SUM(net\_sales)** → values appear quickly.
* Slicers for **Category/Vendor/Region** respond fast (if dims are Import/Dual later).

If you want, say “**share base DAX**” and I’ll give you a small measure pack (Total Sales, Refunds, Net, Return %, YoY, MTD) to finish the first page.

**Yes—make your dimension tables “Composite” by setting them to Dual (usually).**

### Why Dual for dims?

* **Fast slicers/joins:** Dual lets Power BI keep the dim in-memory for filtering while still pushing filters to your DirectQuery facts.
* **Works with Aggregations:** If you add an imported agg table later, dims must be **Import or Dual** for aggregation hits.
* **Best of both:** You get Import-speed interactions without duplicating the fact data.

### What to set

* **Facts:** keep **DirectQuery** (e.g., vw\_fact\_order\_items\_clean, vw\_returns\_clean).
* **Dims (Date, Products, Customer):** set **Storage mode = Dual** (or **Import** if tiny & very static).

### How to change it

Model view → select the table → **Properties (right pane)** → **Advanced** → **Storage mode = Dual** → Apply.

### When to choose ****Import**** instead of Dual for a dim

* The dim is **small (<200k rows)** and **rarely changes**.
* You want absolute max speed and don’t mind scheduling refresh.

### When to ****avoid**** Dual (keep dim in DQ)

* The “dim” is **huge** (behaves like a fact) or updates **many times/hour** and you can’t refresh frequently.
* You’re extremely tight on memory in the Service.

### Keep these guardrails

* Relationships **single direction** from dims to facts.
* Do heavy logic in **SQL views** so folding stays intact.
* If you add an **imported aggregation table**, dims must be **Import/Dual** for it to light up.

**TL;DR:** For your model, set **Date, Products, Customer = Dual**, keep **facts = DirectQuery**. This is the standard high-performance pattern for DQ models.

###### ChatGPT said:

Perfect 💯 absolutelygreen — you’ve already covered the **foundation measures** (Sales, Orders, AOV, Refunds, Rates).

Now let’s shift into **business-problem-driven thinking**. I’ll give you scenarios one by one, you try to think: “what would the business want to know here?”, and only after that we’ll refine into DAX.

## 🔹 Scenario 1: Sales Growth & Trend

👉 Management says: “We want to know if sales are actually growing or just flatlining.”

Questions you, as analyst, should ask:

* Do we compare **current month vs previous month**?
* Or **current year vs last year**?
* Should we look at **growth %** or absolute difference?
* Should we track **cumulative sales (YTD)**?

📌 KPIs you might propose:

* **MoM Growth %**
* **YoY Growth %**
* **Running Total / YTD Sales**

## 🔹 Scenario 2: Customer Insights

👉 CEO says: “Who are our top customers and are they loyal?”

Questions you should ask:

* Define “Top” → by total spend? by orders? by profit?
* Loyalty → repeat purchase rate, retention, churn risk.
* Should we segment customers → New vs Returning?

📌 KPIs you might propose:

* **Top N Customers by Spend**
* **Repeat Purchase Rate** (customers with ≥2 orders)
* **Churn Risk Customers** (no purchase in last 90 days)

## 🔹 Scenario 3: Returns Problem

👉 Ops team says: “Refunds are high in some categories, can we dig deeper?”

Questions:

* Compare **Return % by Category**.
* Identify **Top 5 products with highest refund %**.
* Compare **Region vs Refund %**.
* Vendor-level analysis → which supplier’s products cause more refunds?

📌 KPIs:

* **Return Rate by Category / Product / Region**
* **Refund % vs Overall Avg**

## 🔹 Scenario 4: Logistics & Delivery

👉 Supply Chain team says: “Customers complain deliveries are late.”

Questions:

* Compare **Average Delivery Time** vs SLA (say 3 days).
* Find % orders **delayed >2 days beyond SLA**.
* Region-wise courier performance.

📌 KPIs:

* **Avg Delivery Time**
* **Late Delivery %**
* **Region with Highest Late Deliveries**

# How is this different from PREVIOUSYEAR or SAMEPERIODLASTYEAR?

* **SAMEPERIODLASTYEAR** → takes your **current range** (could be partial) and shifts it back exactly 1 year.
  + Dec 1–15, 2025 → becomes Dec 1–15, 2024.
* **PARALLELPERIOD** → always returns the **full period** (full month/quarter/year).
  + Even if you select Dec 1–15, 2025 → it will return **all Dec 2024**.

**Power BI Report — Pages & Analytical Thinking**

**🔹 Page 1: Executive Summary / Overview**

**Purpose:** Quick snapshot for management → *“Are we winning or losing this month?”*

* **Granularity:** High-level (Company-wide, Current Month/Quarter/Year).
* **KPIs:**
  + Total Sales
  + Total Orders
  + AOV (Avg Order Value)
  + Refund Rate %
  + MoM % & YoY % (trend cards with arrows ↑↓)
* **Visuals:**
  + KPI Cards with indicators
  + Trend line (YTD sales vs PYTD)
  + Donut chart: Refund Rate vs Sales

👉 **Business Decision:** *“Do we need to push sales harder, control refunds, or celebrate growth?”*

**🔹 Page 2: Sales Trends & Growth**

**Purpose:** See how sales are trending over time.

* **Granularity:** Monthly → Quarterly → Yearly.
* **KPIs:**
  + Sales MoM Growth %
  + Sales YoY Growth %
  + Running Totals (YTD, MTD, QTD)
* **Visuals:**
  + Line chart: Sales vs LY
  + Bar chart: MoM % growth
  + Small multiples: Category-wise YoY %

👉 **Business Decision:** *“Which months/quarters are strong, where did we decline?”*

**🔹 Page 3: Customer Insights**

**Purpose:** Know who’s buying and staying.

* **Granularity:** Customer-level.
* **KPIs:**
  + Top N Customers by Spend
  + Repeat Purchase Rate %
  + Churn Risk Customers (no orders last 90 days)
  + New vs Returning Customers
* **Visuals:**
  + Pareto chart (80/20 contribution)
  + Table of top customers (with Refund % & Orders)
  + Line chart: New vs Returning trend

👉 **Business Decision:** *“Do we nurture loyalty, reactivate churn risk, or focus on acquisition?”*

**🔹 Page 4: Product & Category Analysis**

**Purpose:** See which products drive revenue vs refunds.

* **Granularity:** Product → Category → Supplier.
* **KPIs:**
  + Sales by Product/Category
  + Refund Rate by Category/Product
  + Profit Contribution (if margin available)
* **Visuals:**
  + Tree map: Sales by Category
  + Bar chart: Top 10 Products by Refund %
  + Table: Supplier performance

👉 **Business Decision:** *“Which products to promote, fix, or drop?”*

**🔹 Page 5: Returns & Refunds Analysis**

**Purpose:** Focus on leakage → refunds, returns.

* **Granularity:** Category/Region/Customer segment.
* **KPIs:**
  + Refund Rate % (overall + by Category)
  + Refund Amount trend
  + High-risk Customers (≥2 returns/month)
* **Visuals:**
  + Line chart: Refund trend
  + Matrix: Refund % by Category & Region
  + Scatter plot: Refunds vs Sales by Product

👉 **Business Decision:** *“Where are we losing money? Is it operational or customer-behavior driven?”*

**🔹 Page 6: Logistics & Delivery**

**Purpose:** Operations performance.

* **Granularity:** Courier → Region → Order.
* **KPIs:**
  + Avg Delivery Time
  + Late Delivery % (beyond SLA)
  + Region with highest delays
* **Visuals:**
  + Bar chart: Avg Delivery Time by Courier
  + Map: Late deliveries by Region
  + KPI: SLA Breach %

👉 **Business Decision:** *“Do we change courier, optimize routes, or adjust SLA promises?”*

**🔹 Page 7: Operations / Finance View**

**Purpose:** Drill for operations/finance teams.

* **Granularity:** Transaction/order-level.
* **KPIs:**
  + Discount Amount % of Sales
  + Net Sales after Discounts/Refunds
  + Profitability (if margin available)
* **Visuals:**
  + Table: Order-level breakdown
  + Line chart: Net vs Gross Sales
  + Funnel: Orders → Refunds

👉 **Business Decision:** *“Is discounting effective? Are we bleeding profit?”*

**🔹 Granularity Thinking (Analyst Mindset)**

* **Executive (Page 1):** *Company level* (simple, top KPIs).
* **Trends (Page 2):** *Time series* (MoM, YoY, YTD).
* **Customer (Page 3):** *Customer-level* (loyalty, churn).
* **Product (Page 4):** *Product/category-level* (winners vs losers).
* **Returns (Page 5):** *Refund-level* (pain points).
* **Logistics (Page 6):** *Ops-level* (delivery SLA, late%).
* **Finance (Page 7):** *Transactional/profit-level*.

## What else you can add to Page 1

To make it a **complete executive view**:

1. **Waterfall Chart (Net Sales Bridge)**
   * Sales → (-) Discounts → (-) Refunds → Net Sales.
   * Finance & leadership love this because it shows where money leaks.
2. **MoM % Growth Bar (with colors)**
   * Already discussed: green = positive, red = negative.
   * Helps spot months where growth dropped.
3. **Net Sales Trend Line**
   * Line: Total Sales vs Net Sales.
   * Shows the gap caused by refunds & discounts.
4. **Category Contribution %**
   * Stacked bar or donut: % contribution of each category to total sales.
   * “Which categories are actually driving revenue?”
5. **Regional Refund Rate Heatmap (optional)**
   * Map: Refund % by region.
   * Highlights geography-specific issues.

👉 With those, Page 1 becomes a **complete snapshot: Growth + Leaks + Contribution + Regions**.

**📑 Page 3: Customer Insights**

**🎯 Purpose**

* Identify **top customers** (biggest spenders).
* Measure **loyalty vs churn** (repeat vs one-time buyers).
* Spot **at-risk customers** (those who stopped buying).
* Highlight **new vs returning customers**.

**📊 Recommended Visuals & KPIs**

**🔹 KPI Row (top)**

* **Total Customers** = DISTINCTCOUNT(CustomerID).
* **Repeat Purchase Rate %** = customers with ≥2 orders ÷ total customers.
* **Churn Risk Customers** = customers with no purchase in last 90 days.
* **New Customers This Month**.

**🔹 Visuals**

1. **Pareto / Top N Customers (bar chart)**
   * X-axis: Customer Name (Top 10 by Sales).
   * Y-axis: [Total Sales].
   * Tooltip: Refund %, Orders.  
     👉 Shows *who your big accounts are*.
2. **New vs Returning Customers (line or stacked column)**
3. New Customers :=
4. CALCULATE (
5. DISTINCTCOUNT ( Customers[CustomerID] ),
6. FILTER (
7. Customers,
8. CALCULATE ( MIN ( Orders[OrderDate] ) ) = MAX ( 'Date'[Date] )
9. )
10. )
    * Stacked column: New vs Returning by Month.  
      👉 See if growth is driven by acquisition or retention.
11. **Repeat Purchase Rate % (card + trend)**
    * Trend line over months: % of customers with ≥2 orders.  
      👉 Tells if loyalty is improving.
12. **Churn Risk Customers (table)**
    * Show Customer Name, Last Order Date, Total Spend, Refund Rate.
13. Churn Risk :=
14. CALCULATE (
15. DISTINCTCOUNT ( Customers[CustomerID] ),
16. FILTER ( Customers, MAX ( Orders[OrderDate] ) <= TODAY() - 90 )
17. )

👉 Identify customers who stopped ordering.

1. **Customer Lifetime Value (optional)**
   * Avg sales per customer across their lifetime.
   * Useful if management wants long-term loyalty KPIs.

**📝 Page Layout**

-------------------------------------------------

| KPIs: Total Customers | Repeat % | Churn | New |

-------------------------------------------------

| Top Customers (Pareto bar) | Repeat Purchase Trend |

-------------------------------------------------

| New vs Returning (stacked) | Churn Risk Table |

-------------------------------------------------

| (Optional) Customer Lifetime Value card/trend |

-------------------------------------------------

**⚡ Analyst’s Story (how you’d present Page 3)**

* “We served **X customers** this year, with a **repeat rate of Y%**.”
* “Top 10 customers contribute **60% of sales** — we’re highly dependent on them.”
* “**120 customers haven’t ordered in the last 90 days** — they’re at churn risk.”
* “Growth is mainly from **new customers**, but retention is weak.”

**🔹 What is Object Level Security (OLS)?**

OLS = a way to **hide entire objects** (like tables or columns) from specific roles/users in the dataset.

It’s different from **Row Level Security (RLS):**

* **RLS** → filters **rows** of data a user can see (e.g., Sales Rep only sees his region’s data).
* **OLS** → hides **columns or tables** completely (e.g., Finance team sees Profit column, but Sales team doesn’t even know it exists).

**🔹 Example Use Cases**

1. **Hide sensitive financial data**
   * You have a column ProfitMargin.
   * Executives should see it, but Sales team should not.
   * With OLS → Sales team won’t even see that column in fields pane or visuals.
2. **Restrict HR information**
   * HR can see Salary table, but no one else should.
   * OLS hides the Salary table from all non-HR users.
3. **Different dashboards for different audiences**
   * Same dataset powers multiple reports.
   * Using OLS, each audience only “sees” the fields relevant to them.

**🔹 How it works in Power BI**

* You define **Roles** in Power BI dataset.
* Use **Tabular Editor** or XMLA endpoint → set OLS rules.
* Assign users/groups to those roles in Power BI Service.

Example in Tabular Editor:

* Right-click a column → *Set Object Level Security → None/Hidden*.
* Publish dataset → Users in that role won’t even know the column exists.

**🔹 Limitations**

* OLS is only available in **Power BI Premium / Premium Per User (PPU)**.
* Can’t be configured in Power BI Desktop alone → you need **Tabular Editor** or XMLA.
* If a measure depends on a hidden column, you’ll need to secure that too (otherwise users may still infer sensitive info).

✅ In short:

* **RLS** = “which rows can you see?”
* **OLS** = “which columns/tables exist for you?”

###### ChatGPT said:

Bruce here 👋 — got it. You’re talking about **Page 3 of Customers** in your **E-commerce Returns/Refunds + Logistics Delivery project** (your SQL + Power BI project with the Customers table).

Let’s structure **what should go into Page 3 (Customers dashboard/report)** so it’s practical, impactful, and interview-ready.

# 📊 Page 3: Customers

### 1. ****Overview KPIs****

At the top, show customer-centric key metrics:

* **Total Customers**
* **Active Customers (last 90 days)**
* **Repeat Purchase Rate (%)**
* **Churned Customers (no order in last 90 days)**
* **Avg Customer Value (₹ per customer)**
* **Customer Lifetime Value (CLV estimate)**

👉 These KPIs tell at a glance whether your customer base is growing, stable, or declining.

### 2. ****Segmentation & Demographics****

Visuals to break down customers:

* **By City / Country** → Map or bar chart.
* **By New vs. Returning Customers** → Donut chart.
* **By Join Year / Cohort** → Area or line chart (customer growth trend).

👉 Helps identify where customers come from and how cohorts behave.

### 3. ****Behavioral Analysis****

* **Purchase Frequency** (histogram: how many customers order once, twice, 5+ times).
* **RFM Segmentation** (Recency, Frequency, Monetary value):
  + Champions, Loyal, At-risk, New, Churned.
* **Top 10 Customers by Spend** (table with name, city, total spend, last purchase).

👉 Adds interview-style sophistication (RFM is widely asked in analytics).

### 4. ****Returns / Refunds (Customer Lens)****

* **Return Rate per Customer Segment** → which customers return the most.
* **High-risk customers** → customers with ≥2 returns in a month.
* **Refund Amount Contribution** → % of refunds from top customers.

👉 Ties back to your **Returns project** and shows customer quality not just quantity.

### 5. ****Trends Over Time****

* **Monthly New Customers vs. Churned Customers** → stacked line.
* **Revenue by Customer Cohort (first order month)** → cohort heatmap.

👉 Proves you understand **time-based retention analytics**.

### 6. ****Interactivity****

Make Page 3 **interactive**:

* **Slicers/Filters** → Country, City, Customer Segment, Cohort Year.
* **Drill-through** → From customer → order history page.
* **Tooltips** → Hover over customer bar to see lifetime spend & refund %.

### 7. ****Storytelling Insights****

At the bottom, add a text card with **auto-generated insights**:

* “Customer base grew by 12% in last quarter.”
* “10% of customers account for 60% of total revenue.”
* “Refund rates are highest among new customers in Tier-2 cities.”

👉 This ties everything together for management view.

**Dynamic RLS (region-based) — end-to-end**

**0) What you need**

* Dim\_Customers table with a **Region** column (APAC / AMER / EMEA / LATAM).
* Security\_UserRegion table with columns:
  + UPN (viewer’s email / AAD group email)
  + Region (matching your region codes)
* (Recommended) a small unique **Dim\_Region** bridge.

**1) Clean/normalize Region text (Power Query)**

Do this on **both** Dim\_Customers[Region] and Security\_UserRegion[Region] so joins match.

= Table.TransformColumns(PreviousStep, {{"Region", each Text.Upper(Text.Trim(\_)), type text}})

Close & Apply.

**2) Create a unique Region dimension (DAX)**

Dim\_Region = DISTINCT('Dim\_Customers'[Region])

**3) Set relationships (Model view)**

* Security\_UserRegion[Region] (Many) → Dim\_Region[Region] (One)
  + Cross filter **Both**
  + **Apply security filter in both directions: ON**
* Dim\_Region[Region] (One) → Dim\_Customers[Region] (Many)
  + Cross filter **Single** (downstream)
  + Security filter both directions: **OFF**

Why this pattern? Your RLS sits on the tiny security table and safely propagates via the bridge into customers (and from there to facts) without making large tables bidirectional.

*(Alternative without bidirectional: keep both relationships* ***Single****, and add a second role rule on Dim\_Customers using IN VALUES(Security\_UserRegion[Region]). Use this if your org avoids “Both” entirely.)*

**4) Create the RLS role (Desktop)**

Model view → **Manage roles** → **Create** role (e.g., RegionRLS) with **one** rule:

**Table: Security\_UserRegion**

LOWER('Security\_UserRegion'[UPN]) = LOWER(USERPRINCIPALNAME())

*(If you used the “Single + IN” alternative, add a* ***second*** *rule in the same role:*

'Dim\_Customers'[Region] IN VALUES('Security\_UserRegion'[Region])

*)*

**5) Test locally**

* **View as → Other user…** type a UPN that exists in Security\_UserRegion.
* Optional test measures:

Who Am I := USERPRINCIPALNAME()

Visible Regions :=

CONCATENATEX( VALUES('Dim\_Customers'[Region]), 'Dim\_Customers'[Region], ", " )

**6) Publish & assign**

* Publish to workspace.
* Power BI Service → Dataset → **Security** → select RegionRLS → add users or **AAD groups**.
  + Multi-region users: add multiple rows in Security\_UserRegion.
  + “See all” users: add 4 rows (one per region) or add a boolean AllRegions and OR it in the role.

**7) Verify in Service**

* Use **Test as role** on the dataset or open the report as a **Viewer** (admins bypass RLS).
* Confirm visuals only show the allowed regions.

**Incremental Refresh — end-to-end**

**0) What you need**

* Fact tables in **Import** (not DirectQuery).
* A foldable date/time column per fact (e.g., order\_day, return\_day).
* Source supports query folding (Azure SQL is perfect).

**1) Create the required parameters (Power Query)**

**Manage Parameters → New (twice):**

* RangeStart — Type: **Date/Time**, Current value: e.g. 2022-01-01 00:00:00
* RangeEnd — Type: **Date/Time**, Current value: e.g. 2026-01-01 00:00:00

*(Names are case-sensitive and must be exactly RangeStart / RangeEnd.)*

**2) Apply the foldable filter in each fact query**

Make sure the partition column is DateTime **before** filtering. Example for vw\_fact\_order\_items\_clean:

let

Source = Sql.Databases("sqlecom-siddhant.database.windows.net"),

ecom\_dev = Source{[Name="ecom\_dev"]}[Data],

dbo\_vw\_fact\_order\_items\_clean = ecom\_dev{[Schema="dbo", Item="vw\_fact\_order\_items\_clean"]}[Data],

// Ensure DateTime type first

#"Changed Type - order\_day" =

Table.TransformColumnTypes(dbo\_vw\_fact\_order\_items\_clean, {{"order\_day", type datetime}}),

// Incremental filter (exclusive upper bound)

#"Filtered Rows - Incremental" =

Table.SelectRows(#"Changed Type - order\_day", each

[order\_day] >= DateTime.From(RangeStart) and

[order\_day] < DateTime.From(RangeEnd))

in

#"Filtered Rows - Incremental"

(Repeat for returns with return\_day.)

**Check folding:** right-click the last step → **View Native Query** should be enabled.  
If disabled, remove non-folding steps after the filter (e.g., Table.Buffer, complex custom columns) or simplify the view.

**3) Turn on Incremental Refresh (Desktop)**

Model view → select the **fact table** → **Table tools → Incremental refresh & real-time data**:

* **Incrementally refresh this table** ✔
* **Store rows in the last**: *pick a history window*, e.g. **5 Years** (orders) / **3 Years** (returns)
* **Refresh rows in the last**: *pick an incremental window*, e.g. **7 Days** (orders) / **14 Days** (returns)
* **Only refresh complete days**: ✔ (recommended)
* **Detect data changes** (optional but great): choose a LastModifiedAt column if available.
* **Get the latest data in real time with DirectQuery**: optional (Premium/PPU) to create a Hybrid table.

Save the PBIX.

**4) Publish & do the first refresh**

* Publish to a workspace.
* Dataset → **Settings → Data source credentials**: set Azure SQL auth.
* On your Azure SQL server, ensure **“Allow Azure services and resources to access this server”** is **On** (or whitelist the Service’s outbound IPs).
* Click **Refresh now** once. The first run builds all partitions in the “Store” window. Later refreshes only process the “Refresh” window.

**5) Schedule & (if used) Pipelines**

* **Scheduled refresh**: set times (IST) and frequency in Dataset → Settings.
* **Deployment pipelines**: the **policy** moves with the model, but the **physical partitions do not**. After promoting to Test/Prod, run a refresh there to build partitions.

**6) Quick troubleshooting**

* Desktop says “set parameters” → ensure RangeStart/RangeEnd are **parameters** (not queries), exactly named, **Date/Time**, and used in the fact’s filter step.
* Type mismatch errors → cast both sides to DateTime (DateTime.From(...)) or cast the column to Date and use Date.From(...) in the filter.
* Full refresh happening → the filter must use the **same column** you chose in the policy, with >= RangeStart and < RangeEnd. Keep the filter among the last steps.
* Folding off → remove non-folding steps after the filter or move logic into the source/view so the predicate is sargable.

**Suggested defaults for your model**

* **Orders** (vw\_fact\_order\_items\_clean on order\_day): Store **5 Years**, Refresh **7 Days**, Detect changes on modified\_at (if present).
* **Returns** (vw\_fact\_returns\_clean on return\_day): Store **3 Years**, Refresh **14 Days**, Detect changes on updated\_at (if present).

“**RegionRLS**” in **View as** is just the **role you created** for row-level security. Picking it in **Modeling → View as** tells Desktop to **simulate** what a user would see **when that role is applied**.

How to use it (for your dynamic setup):

1. Go to **Modeling → View as**.
2. **Tick** the role **RegionRLS**.
3. Also tick **Other user** and type an email (UPN) that exists in your Security\_UserRegion[UPN] table.
   * This sets what USERPRINCIPALNAME() returns during the simulation.
4. Click **OK** and check your visuals—only that user’s regions (APAC/AMER/EMEA/LATAM) should appear.

Notes:

* **Selecting the role is required**; “Other user” alone won’t apply RLS.
* You can select **multiple roles** to simulate the **union** of their permissions.
* In **Desktop**, this is only a **simulation**. RLS is **actually enforced in the Service** after you publish and assign users/groups to the role.
* Don’t put the **security table** (Security\_UserRegion) on slicers; use region slicers from your **dimension** tables—RLS will limit their values automatically.

# Do this now (OLS in Tabular Editor)

1. **Open TE**  
   Power BI Desktop → **External Tools → Tabular Editor (2/3)**.
2. **Create a dedicated OLS role** (cleaner than reusing your RLS role)
   * In the left tree, right-click **Roles → Create → Role**.
   * Name it e.g. **NoCost** (leave the row filter blank—OLS doesn’t need DAX).
3. **Pick what to hide**
   * In the tree, **Ctrl+click** the sensitive **columns** (e.g., FactOrders[UnitCost], FactOrders[COGS], FactOrders[Margin]).
   * In the **Properties** pane, find **Object Level Security** → for role **NoCost**, set **Value = None**.
   * (If you need to hide an entire table, select the **table** and set OLS for **NoCost** = **None** at the **table** level.)
4. **Save back to Desktop**
   * **Ctrl+S** in TE → switch to Power BI Desktop → **Save** the PBIX.
5. **Publish & assign**
   * Publish to the Service.
   * Workspace → **Datasets → (your dataset) → Security → NoCost** → add users/AAD groups who **must not** see those columns/tables.
6. **Test**
   * In the dataset’s **Security** pane, use **Test as role** (or open the report with a Viewer in that role).
   * Those columns should **vanish** from the Fields list; any visual that depended on them will **error** for that role (so replace those visuals/measures for that audience).

**Page 1 — Ecommerce Dashboard (Overview)**

**What works**

* Strong hero KPIs (Total Sales, AOV, Orders, Refund Rate, Net Sales).
* MoM% bar + Sales vs Last Year line = quick trend + benchmark.
* “Total Sales vs Net Sales” gives a feel for refund impact.

**Tighten up**

* **Dynamic subtitle** on the page (e.g., “YTD Jan–Dec 2025 • Region: All • Category: All”). It reduces “what am I looking at?” confusion.
* **Dual-axis caution** on “Sales vs LastYear”: ensure the secondary axis is clearly labelled or switch to two lines on one axis with data labels at year end.
* **MoM% bars**: add a horizontal zero line + data labels for negative values; consider a 3-month moving average to smooth noise.
* **Refund impact**: add a small **Waterfall** “Gross → Refunds → Net” so execs see the netting logic at a glance.
* **Top 5 categories (refund)**: change pie to **100% stacked bar** with data labels (easier to read) and add a **Drillthrough** to the Returns page.

**Nice to add**

* “**Apply all slicers**” button on top-right for faster multi-filtering.
* A small **KPI Parameter** card row (Sales/Orders/Return%/AOV toggle).

**Page 2 — Sales**

**What works**

* YoY% and MoM% at the top: clear growth signal.
* Rolling 3M chart: good for momentum.
* Category area + month grid table: breadth + detail.

**Tighten up**

* **Best/Worst month** cards: add a **variance value** underneath (e.g., “+$92K vs prior month”).
* **Orders by Quarter** drops to zero in Q4 in your screenshot—verify if that’s a data gap, a slicer selection, or an end-of-year issue (Incremental Refresh window, Date table coverage, or filters). If gap is real, annotate it.
* **Category stack area** can hide small categories. Add **Small multiples** (one per top category) or switch to **top N + others**.

**Nice to add**

* **Variance Waterfall**: “Why are we down/up YoY?” (by Category/Region).
* **Bookmark pair**: “Growth view” (YoY stacks + waterfall) vs “Seasonality view” (rolling 3M + monthly).

**Page 3 — Customers**

**What works**

* Good mix of base size (Unique Customers), activity (Active 3M), and quality (Repeat%, ACV).
* Churn vs Active overlay is insightful.

**Tighten up**

* The large empty placeholder—use it for a **Cohort Retention heatmap** (acquired month vs month-n retention). This is the single most useful customer visual for ops.
* Define **Active Customers3M** in a tooltip (hover) so meaning is obvious: “distinct customers with ≥1 order in the last 90 days.”
* **Repeat customers (>=2)**: add a small card **Repeat Revenue Share %** (what portion of sales comes from repeat customers).

**Nice to add**

* **RFM segmentation** (Recent, Frequency, Monetary) as a matrix or clustered column.
* **Drillthrough**: Customer profile (orders, returns, lifetime value, last purchase, region).

**Page 4 — Returns**

**What works**

* You’ve planned the right slots: trends, reasons, vendor vs timing.

**Tighten up (fill the blanks)**

* Define 5 returns KPIs at the top:
  + **Return Rate %** (Units Returned / Units Sold)
  + **Refund Amount** (currency)
  + **Avg Days to Return**
  + **Avg Days to Refund** (SLA)
  + **Within-Policy %** (returns within allowed window)
* **Trend**: show **Return Rate** alongside **Units Returned** (secondary axis or separate card) to avoid volume vs rate confusion.
* **Reasons**: keep it sorted desc; add **% of total** labels; enable **drillthrough** to Product/Vendor detail.
* **Vendor x Return/Refund days**: use a **clustered column + line** or **box plot** (via custom visual) to catch SLA outliers. Add a **Pareto** (top vendors causing 80% of refunds).

**Nice to add**

* **Heatmap**: Category × Reason by Refund Amount.
* **Playbook section** (text box): “Actions this month” (e.g., tighten packaging for top 3 SKUs, vendor escalation).

**Page 5 — Products**

**What works**

* Clear cards: Top category, subcategory, top product by revenue, ASP.
* Treemap gives a quick share-of-sales read; product table is handy for drill.

**Tighten up**

* Add a **search** (text filter) for product\_name; product analysis needs easy lookup.
* The blank visual—fill with **Top N driver** (Top 20 SKUs by Sales/Return%), with a **TopN slicer** (5/10/20).
* The line “Sales by Month and category” collapses late in the year—confirm data coverage or filters (similar note as Page 2).

**Nice to add**

* **ABC Classification** (A=80% sales, B=15%, C=5%) as a bar with thresholds.
* **Return Rate overlay** on product table (conditional formatting) to flag good sellers with high returns.
* **Price elasticity What-If**: a slider adjusting ASP ±10% and a measure showing expected revenue change.

**Cross-page polish**

**Slicers & Parameters**

* Add **Axis Parameter** (Region / Category / Subcategory / Brand) on Sales & Products pages. You already show an AxisP… slicer—great; wire it into key charts.
* Add **KPI Parameter** on the Overview for a compact KPI bar that users can toggle.
* Put **“Apply all slicers”** and **“Clear all”** buttons on each page; keep the “Reset” bookmark up top (you already have it).

**Design & accessibility**

* Unify number formats (K/M/B, thousands separators) and DAX measure formats (percentage with 1 decimal where needed).
* Use consistent color semantics (e.g., negative MoM% in a warning color, positive in a success color).
* Ensure contrast on purple KPI cards; add brief captions under charts to explain the “so what”.

**Performance**

* You’ve set **Incremental Refresh** (great). Add **Aggregation tables** if queries hit large grain (e.g., product-day) and pages feel heavy.
* On DirectQuery or heavy visuals, use **Optimize → Query reduction** preset and rely on **Apply all slicers**.
* Prefer **measure-driven** calcs over calculated columns for dynamic logic.

**RLS / OLS**

* Your Region RLS is set—test via “View as” and in Service with a Viewer account.
* If you plan a partner version, add OLS (hide UnitCost, COGS, Margin) and swap margin visuals for revenue alternatives on partner pages.

**High-impact DAX snippets (drop-in)**

-- YoY% (generic)

YoY % :=

VAR Curr = [Total Sales]

VAR Prev = CALCULATE([Total Sales], DATEADD('Date'[Date], -1, YEAR))

RETURN DIVIDE(Curr - Prev, Prev)

-- MoM% (generic)

MoM % :=

VAR Curr = [Total Sales]

VAR Prev = CALCULATE([Total Sales], DATEADD('Date'[Date], -1, MONTH))

RETURN DIVIDE(Curr - Prev, Prev)

-- Rolling 3M Sales

Sales Rolling 3M :=

CALCULATE([Total Sales], DATESINPERIOD('Date'[Date], MAX('Date'[Date]), -3, MONTH))

-- Return Rate %

Return Rate % :=

DIVIDE([Units Returned], [Units Sold])

-- Discount scenario

Sales After Discount :=

VAR d = 'Discount %'[Discount % Value]

RETURN [Total Sales] \* (1 - d/100.0)

**Suggested “next 10” checklist**

1. Add **dynamic subtitle** and **Apply all slicers** on each page.
2. Fix any data gaps late in the year (Date table coverage, filters, refresh window).
3. Replace refund pie with 100% stacked bar; add **Refund Waterfall** on Overview.
4. Add **Variance Waterfall (YoY)** on Sales and annotation for anomalous months.
5. Add **Cohort heatmap** + **Repeat Revenue Share** on Customers.
6. Build top-row **Returns KPIs** and SLA charts; add vendor drillthrough.
7. Enable **product search**, **TopN slicer**, **ABC** on Products; style table with conditional format on Return%.
8. Add **KPI** and **Axis** field parameters where useful.
9. Standardize number formats & color semantics across pages.
10. (Optional) Add **Aggregation tables** if any page feels slow.

# 1) Lock in refresh & reliability

1. **Credentials**

* Workspace → **Datasets** → ecommerce semantic model → **Settings → Data source credentials** → must show **Valid**.
* **Gateway connection**: for Azure SQL (public) leave **Not configured** (no gateway). Only map a gateway if your source is on-prem/private.

1. **Schedule refresh**

* Same page → **Scheduled refresh** → **On** → pick **time zone & times** (stagger runs if you have multiple datasets).
* Keep **Allow users to refresh this dataset** off unless you want self-service refresh.

1. **Notifications**

* **Refresh failure notifications** → add yourself + owner list.
* (Optional) **Subscriptions**: open your report → **Subscribe** to receive daily/weekly screenshots/PDF.

# 2) Security: RLS/OLS in the Service

1. **RLS assignments**

* Dataset → **Security** → open role RegionRLS → add **users or AAD groups**.
* **Test as role** (top right) with a few emails.

1. **OLS (if you created it in Tabular Editor)**

* Assign users to the OLS role (e.g., NoCost).
* Open the report as a Viewer in that role—sensitive columns should disappear.

# 3) Publish an App (the shareable package)

* Workspace → **Create app**.
* **Audiences**: e.g., Execs, Regional Managers, Partners (each can see a different subset of pages).
* **Navigation**: group pages, add descriptions/tooltips, set a **landing page**.
* **Publish** → share the App link with stakeholders.

# 4) Deployment Pipelines (Dev → Test → Prod)

1. **Workspaces**: create **Dev/Test/Prod** workspaces (use Premium/PPU if you need XMLA write, larger models, or hybrid tables).
2. **Create pipeline**: **Deployment pipelines → New pipeline** → name it → **Assign** your **Dev** workspace.
3. **Deploy** to Test, then to Prod.
4. **Stage rules** (very important):
   * **Data source rules**: point each stage to the correct server/DB.
   * **Parameter rules**: map any M parameters that differ by stage.
5. **Per stage**: open the dataset → **Data source credentials** → sign in once; **Refresh now** (first run builds partitions).
6. When ready, **Update App** in each stage (or only in Prod) for users.

# 5) Monitoring & ops

* **Usage metrics**: Report → **View usage metrics report**. Watch views, unique viewers, average time.
* **Capacity watch** (Premium/PPU): **Workspace → Settings → Premium** shows refresh/query wait times & memory. Consider **auto-scale** if needed.
* **Audit** (tenant admins): **Admin portal → Audit logs** to track sharing, exports, etc.

# 6) Performance & UX polish (quick wins)

* On heavy pages, add **Apply all slicers** button and use **Optimize → Query reduction** (keeps visuals from re-querying on every tweak).
* Pin the most important visuals to a **Dashboard** and (optionally) add **tile alerts** (alerts work on dashboard tiles).
* Standardize formats (K/M/B; one decimal on %), add a **dynamic subtitle** on each page showing active Year/Region/Category filters.
* If any page still feels slow: consider **aggregations** (e.g., product-month), remove unused columns, and prefer measures over calc columns.

# 7) Governance & trust

* **Endorsement**: Dataset → **Settings → Endorsement** → mark **Promoted/Certified** (if your org uses certification reviews).
* **Sensitivity labels**: apply labels required by your data policy.
* **Access control**: in each workspace, assign people as **Viewer** unless they truly need edit rights (Admins/Contributors bypass RLS).

# 8) Source control (Git) with PBIP

* In Desktop: **File → Save As → Power BI project (.pbip)**.
* Put the PBIP folder in a repo (git init, commit, push).
* Use branches/PRs for changes; **publish from Dev branch**; then deploy via the pipeline.
* For model changes made in Tabular Editor, **save back into the PBIX/PBIP** before publishing so everything’s tracked.

# 9) Your specific backlog (based on our build)

* **Incremental Refresh**: already defined—keep an eye on **Refresh history**; expect long first run, shorter incrementals after.
* **Field parameters**: add **KPI switcher** and **Axis switcher** across key pages.
* **What-If sliders**: at least **Discount %** and **Trailing days** on Sales/Products.
* **Returns page**: add top KPI row (Return Rate, Refund Amount, Days to Return/Refund, Within-Policy %), vendor SLA chart, and drillthrough to product/vendor.
* **Customers page**: add **cohort heatmap** and **Repeat revenue share**; optional **RFM** segmentation.
* **Products page**: add **search**, **Top-N** slicer, **ABC** classification, and conditional formatting on return rate.
* **App audiences**: create tailored views for Execs/Regions/Partners (hide cost/margin visuals for OLS users).

That message means you’re trying to add an email that isn’t in your tenant. You can only add:

1. **Internal users** (same tenant), or
2. **External “guest” users** that have been invited into your tenant via Microsoft Entra ID (Azure AD). “Random” emails that aren’t guests will trigger the warning and won’t work.

Here’s how to do it properly for an external person like apac.analyst@yourco.com.

**Add an external (guest) user the right way**

1. **Enable external sharing** (admin step)  
   Power BI Admin portal → **Tenant settings → External sharing** must allow sharing with external users/B2B guests (optionally limited to specific domains). [Microsoft Learn](https://learn.microsoft.com/en-us/fabric/enterprise/powerbi/service-admin-entra-b2b?utm_source=chatgpt.com)
2. **Invite them as a guest to your tenant** (admin or anyone allowed by Entra settings)  
   Microsoft Entra admin center → **Users → New user → Invite external user** → enter their email → send invite. They must **accept** the invitation. [Microsoft Learn+1](https://learn.microsoft.com/en-us/entra/external-id/external-collaboration-settings-configure?utm_source=chatgpt.com)
3. **License**
   * If the workspace is **Premium capacity**, guests can view with a **Free** license; otherwise guests need **Pro or PPU** (you can assign a Pro/PPU from your tenant to the guest). [Microsoft Learn](https://learn.microsoft.com/en-us/fabric/enterprise/powerbi/service-admin-entra-b2b?utm_source=chatgpt.com)[Microsoft Fabric Community](https://community.fabric.microsoft.com/t5/Service/How-to-allow-Guest-Users-to-use-our-Power-BI-pro-license-instead/m-p/2658624?utm_source=chatgpt.com)
4. **Add them to your dataset role**  
   Power BI Service → Workspace → **Datasets → … → Security** → open your role (e.g., RegionRLS or NoCost) → search and **add the guest user** (or better, add an **AAD security group** and put the guest in that group). [Microsoft Learn](https://learn.microsoft.com/en-us/fabric/security/service-admin-row-level-security?utm_source=chatgpt.com)
5. **Make sure RLS mapping matches their guest UPN**  
   For B2B users, USERPRINCIPALNAME() often looks like firstname\_lastname#EXT#@yourtenant.onmicrosoft.com. Put **that exact value** in your Security\_UserRegion[UPN] mapping (or lowercased if you’re comparing with LOWER()). You can see the guest’s UPN in Entra user details, or create a card measure Who Am I = USERPRINCIPALNAME() and ask them to open the report once invited. [Microsoft Learn](https://learn.microsoft.com/en-us/fabric/security/service-admin-row-level-security?utm_source=chatgpt.com)

**If you want to add many people at once**

* Create an **Entra security group** (e.g., pbi-ecommerce-viewers), add users/guests to it, then add the **group** to the dataset role. You still need the **user-level rows** in Security\_UserRegion for dynamic RLS, or have an ETL that expands group members into user rows nightly. [Microsoft Learn](https://learn.microsoft.com/en-us/entra/external-id/external-collaboration-settings-configure?utm_source=chatgpt.com)

It depends which “admin access control” you mean. Here are the three places people usually look, with exact click-paths:

# 1) Power BI ****Admin portal**** (tenant-wide settings like external sharing, exports, etc.)

Only **Power BI/Fabric admins** see this.

* Go to **app.powerbi.com** → top-right **⚙️ (Settings)** → **Admin portal**.
* In the left pane you’ll see:
  + **Tenant settings** → (e.g., External sharing, Export/Download, Content sharing).
  + **Capacity settings**, **Workloads**, **Usage metrics**, etc.

If you don’t see **Admin portal**, you’re not a Power BI admin. Ask your org’s Fabric/Power BI admin to adjust Tenant settings (e.g., allow external B2B) or to grant you admin rights.

# 2) Power BI ****workspace access**** (who is Admin/Member/Contributor/Viewer for a workspace)

* Open your **workspace** → top-right **Access** (or **More options … → Workspace access**).
* Here you control **who can edit or view**, and their role:
  + **Admin** / **Member** / **Contributor** / **Viewer**.

# 3) ****Dataset security**** (RLS/OLS role assignments on your semantic model)

* Workspace → **Datasets + dataflows** → hover **ecommerce semantic model** → **… (More options)** → **Security**.
* Open each role (e.g., RegionRLS, NoCost) → **Add members** (users or Entra ID groups) → **Save**.
* Use **Test as role** (top-right) to simulate.

# (Related) Microsoft Entra ID (Azure AD) ****groups & guests****

If you need to create a security group or invite a guest user:

* Go to **entra.microsoft.com** (Microsoft Entra admin center).
  + **Groups** → **New group** (create a **Security** group; add members).
  + **Users → New user → Invite external user** (B2B guest).
  + **External Identities → External collaboration settings** (controls guest behavior—admin level).

# Quick “which one do I use?” map

* **Change org-wide rules** (allow external/B2B, exports, embed): **Admin portal → Tenant settings**.
* **Give someone access to a workspace/report**: **Workspace → Access**.
* **Assign users to RLS/OLS roles**: **Dataset → … → Security**.
* **Create/manage groups and guests**: **Entra admin center** (Groups/Users).