**Overview**

The five scripts together create a small analytical data-mart and a set of reusable SQL assets for inventory, pricing, product and store performance analysis. Run them in the order shown below.

**1. sqlcode.sql — Schema & ETL Foundations**

**Purpose**

* Creates inventory\_sql\_project schema and loads a star-like structure around the fact table sales.
* Provides the base objects that all other scripts query.

**Key steps**

* Dimension tables: products, stores, dates, seasons (manual month-to-season map).
* Fact table: sales (PK = date, store\_id, product\_id) with FKs to each dimension.
* Post-load feature engineering:  
  – inventory\_lvl\_at\_eod (ending stock)  
  – revenue = price \* (1 – discount/100) \* units\_sold (handles % discount)  
  – deviation\_from\_forecast and diffr\_from\_comp\_price helper columns.

Run this script first; it seeds every subsequent view/query.

**2. inventoryopt.sql — Inventory Optimisation Views & Dashboard**

**Analytical views**

* reorder\_points – ROP = 7 × average daily demand.
* safety\_stock – 2 × std-dev of demand.
* economic\_order\_quantity – Classic EOQ using fixed S = ₹50, H = ₹10.
* service\_level – % of days demand was fulfilled (fill-rate).
* stockout\_rate – % of days inventory < demand.
* inventory\_turnover – Sales ÷ average stock.
* abc\_classification – Revenue-based Pareto segmentation (A/B/C).

**Diagnostic queries**

* “Products below ROP”, “Inventory age tracking” and “Most recent stock change”.2

**Dashboard**  
inventory\_dashboard joins all views for a one-row-per-product scorecard; the final SELECT sorts by highest stock-out risk.2

Run after sales\_data is available (created through sqlcode.sql).

**3. pricing-and-promo-analysis.sql — Pricing & Promotion Insights**

**Discount performance**

* Bucketed discount ranges (0 %, 1–5 %, 6–10 %, 11–15 %, 16–20 %) with average units sold / revenue and event counts.
* Same logic split by store\_id and by product\_id for granular visibility.

**Price competition**

* Average unit movement versus differential to competitor price (“Much Cheaper” … “Much Expensive”).

**Elasticity notes**  
A comment reminds analysts to estimate price elasticity via external regression tooling if the SQL engine lacks built ins.

**4. product-lvl-analysis.sql — Product-Centric Analytics**

**Revenue concentration**

* Contribution % by product and by category; highlights top-/bottom-performers.

**Category normalisation**

* Normalised revenue per product inside each category to control for assortment size.

**Regional splits**

* Revenue share by region, by region-product and region-category.

**Movement speed**

* NTILE(5) ranking of average units sold (“Fast-moving”, “Moderate”, “Slow-moving”).

**Seasonality**

* Product-season and Category-season sales using the seasons dimension.

**Quick winners**

* Query for “top 5 products per region” by units sold.

**5. store-lvl-analysis.sql — Store-Level Performance**

**Core KPIs**

* Total / average revenue and units sold per store.
* Cross-tabbed monthly revenue for 2022-2023 (calendar heat-map source).
* Average inventory turnover computed from daily stock & sales.

**Stock health**

* Counts under-stock events (inventory\_level < units\_sold) and estimated lost units/revenue.

**Growth & composite scoring**

* YoY revenue growth per store (2022 → 2023).
* Weighted performance score: 50 % revenue, 30 % turnover, 20 % (inverse) under-stock events.

**Execution Order**

1. sqlcode.sql – build schema, dimensions, fact table and derived columns.
2. inventoryopt.sql – create inventory optimisation views/dashboard.
3. Run analytical scripts in any order:  
   – pricing-and-promo-analysis.sql  
   – product-lvl-analysis.sql  
   – store-lvl-analysis.sql

All scripts assume the default database is inventory\_sql\_project and that ANSI SQL functions (e.g., STDDEV, window functions) are supported.