Solving Inventory Inefficiencies Using SQL

Prepared by: Meet Jain & Muskan Verma

1. 🧩 Introduction: The Business Context

Urban Retail Co., a fast-growing retail chain operating across both physical and online stores, manages over 5,000 SKUs with diverse sales behavior across multiple regions. However, increasing operational complexity, lack of integrated data intelligence, and manual decision-making have made inventory management inefficient.

Key challenges included:

- Frequent stockouts of fast-moving products
- Overstocking of slow-moving items
- Lack of real-time insights into demand and performance
- Poor visibility across product categories, regions, and suppliers

📌 Our project aims to use SQL-driven analytics and KPI dashboards to detect inefficiencies, optimize inventory behavior, and guide smarter business decisions.

2. 🎯 Mission Fulfilled: How We Solved the Problem

We successfully executed a complete SQL-driven monitoring and optimization pipeline that aligned with all expected deliverables:

Expectation	What We Did
Scalable, clean SQL queries	Delivered modular, documented, and optimized queries
Diagnosed inefficiencies	Identified under/overstocking, inventory lags, low turnover, forecast errors
Delivered actionable insights	Stock adjustment logic, KPI dashboards, discount sensitivity analysis
Creativity beyond brief	Added bonus queries, anomaly detection, trend analyses
Communicated findings clearly	Created a complete report, documentation, and interactive dashboard

3. 🛠️ Methodology & Design Rationale

📌 Operational Case Study: Detecting Inventory Anomalies Due to Lag in Delivery Logging

During data profiling, we observed unexplained inventory "jumps" (e.g., S002 - P0016 jumped from 98 → 401 units in one day). These spikes were not aligned with sales or order behavior.

🔎 We concluded that such events likely stem from unlogged replenishments or warehouse dispatch delays not captured in real time. This causes distortions in daily-level inventory KPIs and creates forecasting mismatches.

✅ Solution Proposed:

- Add delivery confirmation logic in future data pipelines
- Apply buffer or lag-aware modeling for real-time analytics

📌 Reorder Point Estimation with Inventory-Aware Capping

Initially, our calculated ROP = 3 × 7-day average demand often exceeded 300–400 units due to high sales spikes. Meanwhile, actual inventory rarely crossed 120 units.

🚫 Result: All SKUs were flagged as low-inventory for 730 days — making the metric unusable.

✅ Fix Applied:

- Recalculated demand over 730-day average
- Applied: LEAST(ROP, 120 units)

This aligned ROP with operational limits, restoring usefulness to low_inventory_days and stock adjustment logic.

📌 Stockout Detection: An Empty Signal is Still a Signal

We ran a stockout detection query where:

WHERE units_sold > 0 AND inventory_level = 0

🟢 No stockouts were found — either due to:

- Strong supply chain controls
- Clean simulation data

✅ Noted as: "No stockouts detected. This itself is a KPI and indicates inventory was never fully depleted during sales activity."

4. 📈 Core KPIs & Analytical Approach

We structured our KPIs to reflect essential dimensions of inventory performance:

Metric	KPI Description	SQL Script File
Avg Inventory	Inventory available per SKU across time	avg_inventory_level.sql
Inventory Turnover	Sales velocity vs inventory on hand	inventory_turnover_analysis.sql
Reorder Point	Estimated minimum inventory before reordering is triggered	reorder_point_estimation.sql
Low Inventory Days	# of days inventory < ROP — risk of stockout	kpi_low_inventory_days.sql
Stockout Days	Days when units_sold > 0 but inventory_level = 0	stockout_days.sql (no stockouts)
Forecast Accuracy	Absolute error between actual and predicted sales	forecast_error_analysis.sql
Stock Recommendations	Actionable suggestions based on movement, risk, and stock size	recommend_stock_adjustments.sql

5. 🔁 Recommended Stock Adjustments

This table is the strategic output of our entire analysis. It guides inventory decision-making using SQL-powered intelligence.

🧠 Logic:

- Used avg_inventory, turnover_ratio, and low_inventory_days
- Applied window functions (NTILE / PERCENT_RANK)
- Flagged SKUs as Increase / Reduce / Balanced

📊 Table Sample:

store_id	product_id	avg_inventory	turnover_ratio	low_inventory_days	stock_recommendation
S001	P0017	92.0	0.8	344	Increase
S002	P0056	120.0	0.3	421	Reduce
S003	P0033	87.2	1.1	120	Balanced

📌 Recommendation thresholds (data-driven):

Metric	Threshold Strategy
High Inventory	Top 25% of avg_inventory
Low Turnover	Bottom 25% of turnover_ratio
Frequent Low Inv. Days	Top 25% of low_inventory_days

6. 🎯 Mission Fulfilled: How We Solved the Problem

We successfully executed a complete SQL-driven monitoring and optimization pipeline that aligned with all expected deliverables:

Expectation	What We Did
Scalable, clean SQL queries	Delivered modular, documented, and optimized queries
Diagnosed inefficiencies	Identified under/overstocking, inventory lags, low turnover, forecast errors
Delivered actionable insights	Stock adjustment logic, KPI dashboards, discount sensitivity analysis
Creativity beyond brief	Added bonus queries, anomaly detection, trend analyses
Communicated findings clearly	Created a complete report, documentation, and interactive dashboard

7. 🧱 Database Optimization

We satisfied both optimization mandates listed in the deliverable:

✅ **Normalization & Schema Design**

- Converted flat data into a relational star schema
- Created product_master, store_master, and sales tables
- Created an ERD using dbdiagram.io and explained foreign key relationships
- Sales table structured as a fact table joined with store and product dimensions

✅ **Performance Indexing**

- Indexes created on:
    - store_id, product_id
    - date
    - composite keys for join optimization
- Isolated this logic in **schema_optimization.sql**

📌 These steps ensured efficient query execution and scalability for large data volumes.

8. 🗂️ SQL Queries: Core + Extended

🔹 As Required in Problem Statement

Query Name	Description
stock_level_calculations.sql	Tracked inventory levels daily per store/product
low_inventory_days.sql	Identified days when inventory < ROP
reorder_point_estimation.sql	Estimated reorder threshold using moving average demand and lead time cap
inventory_turnover_analysis.sql	Compared average inventory to units sold ratio
summary_kpis.sql	Generated aggregate metrics for dashboard use

🔹 Additional Creative Queries

Query Name	Purpose & Insight
stockout_days.sql	Detected days with 0 inventory but positive sales — result: no stockouts found
discount_vs_sales.sql	Showed positive correlation between discount and average units sold
forecast_error_analysis.sql	Compared forecast vs actual sales with % error (MAPE-style analysis)

Query Name	Purpose & Insight
inventory_jump_detection.sql	Detected unrealistic inventory jumps → identified lagged warehouse logging
recommend_stock_adjustments.sql	Used percentile-based logic for stock optimization suggestions

9. 📈 KPI Dashboard Elements

- **Low Stock Count (Card)**
    - Displays the number of store-product pairs that are currently below their calculated reorder point.
    - Helps identify stockout risks and prioritize replenishment actions.
- **Top 5 Selling Products (Card + Clustered Bar Chart)**
    - Highlights the highest-selling SKUs across the dataset.
    - Provides quick visibility into fast-moving items to inform inventory planning and promotional focus.
- **Overstocked Count (Card + Table + Slicer)**
    - Identifies products that are overstocked using percentile thresholds (e.g., high inventory + low turnover).
    - Enables deep-dive into overstocked SKUs via an interactive table and slicers to filter by store or category.
- **Forecast Accuracy (Gauge Chart + Card + Table + Slicer)**
    - Measures average forecast error per product-store pair using actual vs. forecasted sales.
    - Allows teams to evaluate prediction reliability and improve demand planning.

10. 📌 Impact Summary: Business Value of Our Solution

By leveraging SQL-driven inventory analysis, our project directly enables Urban Retail Co. to:

Business Impact Target	How We Addressed It
✅ Smarter inventory decisions	Derived reorder points, risk flags, and stock suggestions using SQL KPIs
✅ Reduce stockouts and overstocks	Analyzed both extremes through turnover, ROP, and stock movement
✅ Improve supply chain efficiency	Detected delivery lags and cleaned forecasting logic
✅ Enhance customer satisfaction & profitability	Ensured data-informed reordering and sales optimization