## TITLE: Elevate Customer Satisfaction: Revolutionize Supply Chain with SQL-Driven Inventory Optimization

## RATIONALE FOR THE PROJECT

## Inventory optimization is the process of strategically managing inventory levels to maximize efficiency

## and minimize costs while ensuring that customer demand is met. It involves analyzing factors such as demand patterns, lead times, supply chain constraints, and storage costs

## to determine the optimal quantity of each item to stock.

## The goal of inventory optimization is to strike a balance between having enough inventory to fulfill customer orders promptly while avoiding excess stock that ties up capital and increases carrying costs.

## Implementing a comprehensice inventory optimization system powered by MySQL is imperative for TechElectro INc. due to several compelling reasons.

## Cost Reduction

## Enhanced Customer Satisfaction

## Competitive Advantage

## Profitability

## AIMS OF THE PROJECT

## The primary goals of this project are to implement a solid inventory optimization system using MySQL and address business challenges effectively

## Optimal Inventory levels

## Data\_driven Decisions

## Prelim-creation of Schema/Database

CREATE SCHEMA tech\_electro;

USE tech\_electro;

## DAT exploration

SELECT \* FROM tech\_electro.external\_factors limit 10;

SELECT \* FROM tech\_electro.inventory\_data limit 10;

SELECT \* FROM tech\_electro.product\_information limit 10;

SELECT \* FROM tech\_electro.sales\_data limit 10;

## Understanding the structure of the datasets

DESCRIBE tech\_electro.product\_information;

DESC tech\_electro.sales\_data;

## Data Cleaning

## Changing to the right data type for all columns

## external factors table

## SalesDate DATE, GDP Decimal (15,2), InflationRate Decimal (5,2), Seasonalfactor decimal (5,2)

Alter Table tech\_electro.external\_factors

Add column New\_Sales\_Date DATE;

SET SQL\_SAFE\_UPDATES = 0; ##Turning off safe updates

Update tech\_electro.external\_factors

SET New\_Sales\_Date = STR\_TO\_DATE(Sales\_Date,'%d/%m/%Y');

Alter Table tech\_electro.external\_factors

Drop column Sales\_Date;

Alter TABLE tech\_electro.external\_factors

Change Column New\_Sales\_Date Sales\_Date DATE;

Select \* from tech\_electro.external\_factors;

Alter TABLE tech\_electro.external\_factors

Modify Column GDP Decimal (15,2);

Alter TABLE tech\_electro.external\_factors

Modify Column Inflation\_Rate Decimal (5,2);

Alter TABLE tech\_electro.external\_factors

Modify Column Seasonal\_Factor Decimal (5,2);

Show columns from tech\_electro.external\_factors;

## Product data

## Product\_ID INT NOT NULL, Product\_Category TEXT; Promotions ENUM ('yes', 'no')

Alter TABLE tech\_electro.product\_data

Add column NewPromotions ENUM ('yes', 'no');

Update tech\_electro.product\_data

Set NewPromotions = CASE

When Promotions = 'yes' Then 'yes'

When Promotions = 'no' then 'no'

Else Null

End;

Alter TABLE tech\_electro.product\_data

drop column Promotions;

Alter TABLE tech\_electro.product\_data

Change column NewPromotions Promotions ENUM ('yes', 'no');

describe tech\_electro.product\_data;

## Sales Data

Alter Table tech\_electro.sales\_data

Add column New\_Sales\_Date DATE;

Update tech\_electro.sales\_data

SET New\_Sales\_Date = STR\_TO\_DATE(Sales\_Date,'%d/%m/%Y');

Alter Table tech\_electro.sales\_data

Drop column Sales\_Date;

Alter TABLE tech\_electro.sales\_data

Change Column New\_Sales\_Date Sales\_Date DATE;

desc tech\_electro.sales\_data;

## Identifying missing values using 'IS NUll' function

## external factor data

Select

SUM(CASE WHEN Sales\_Date IS NULL THEN 1 ELSE 0 END) As missing\_sales\_date,

SUM(CASE WHEN GDP IS NULL THEN 1 ELSE 0 END) As missing\_gdp,

SUM(CASE WHEN Inflation\_Rate IS NULL THEN 1 ELSE 0 END) As missing\_inflation\_rate,

SUM(CASE WHEN Seasonal\_Factor IS NULL THEN 1 ELSE 0 END) As missing\_seasonal\_factor

FROM

tech\_electro.external\_factors;

## Identifying missing values using 'IS NUll' function

## Product\_data

Select

SUM(CASE WHEN Product\_ID IS NULL THEN 1 ELSE 0 END) As missing\_product\_ID,

SUM(CASE WHEN Product\_Category IS NULL THEN 1 ELSE 0 END) As missing\_product\_category,

SUM(CASE WHEN Promotions IS NULL THEN 1 ELSE 0 END) As missing\_promotions

FROM

tech\_electro.product\_data;

## Identifying missing values using 'IS NUll' function

## Sales\_data

Select

SUM(CASE WHEN Product\_ID IS NULL THEN 1 ELSE 0 END) As missing\_product\_ID,

SUM(CASE WHEN Sales\_Date IS NULL THEN 1 ELSE 0 END) As missing\_sales\_date,

SUM(CASE WHEN Inventory\_Quantity IS NULL THEN 1 ELSE 0 END) As missing\_inventory\_quantity,

SUM(CASE WHEN Product\_Cost IS NULL THEN 1 ELSE 0 END) As missing\_product\_cost

FROM

tech\_electro.sales\_data;

## Identifying using 'Group BY' and 'Having' clauses and remove them if necessary

## external factor data

Select Sales\_Date, Count(\*) AS count

From tech\_electro.external\_factors

Group by Sales\_Date

Having count>1;

Select Count(\*)

From

(Select Sales\_Date, Count(\*) AS count

From tech\_electro.external\_factors

Group by 1

Having count>1) as Duplicate;

## Product data

Select

Product\_ID, Product\_Category, Count(\*) AS count

From tech\_electro.product\_data

Group by 1,2

Having count>1;

Select Count(\*)

From

(Select

Product\_ID, Product\_Category, Count(\*) AS count

From tech\_electro.product\_data

Group by 1,2

Having count>1) as Duplicate;

## sales data

Select

Prod\_ID, Sales\_Date, Count(\*) AS count

From tech\_electro.sales\_data

Group by 1,2

Having count>1;

## Dealing with duplicates for external\_factors and Product\_data

## external factor

Delete e1 from tech\_electro.external\_factors e1

INNER JOIN (

Select Sales\_Date,

ROW\_NUMBER() OVER (PARTITION BY Sales\_Date ORDER BY Sales\_Date) AS rn

FROM tech\_electro.external\_factors

) e2 ON e1.Sales\_Date = e2.Sales\_Date

WHERE e2.rn>1;

## Product\_data

Delete p1 from tech\_electro.product\_data p1

INNER JOIN (

Select Product\_ID,

ROW\_NUMBER() OVER (PARTITION BY Product\_ID ORDER BY Product\_ID) AS rn

FROM tech\_electro.product\_data

) p2 ON p1.Product\_ID = p2.Product\_ID

WHERE p2.rn>1;

##Data Integration

## Combine Sales\_data and Product\_data first

## New\_Sales\_Date should be used

DROP VIEW IF EXISTS Sales\_Product\_Data;

-- Alternatively, if it's a table, use:

-- DROP TABLE IF EXISTS Inventory\_Date;

CREATE VIEW Sales\_Product\_Data AS

Select

s.Prod\_ID,

s.Sales\_Date,

s.Inventory\_Quantity,

s.Product\_Cost,

p.Product\_ID,

p.Product\_Category,

p.Promotions

from sales\_data s

Join product\_data p On s.Prod\_ID=p.Product\_ID;

Select \* from sales\_data

Select \* from Sales\_Product\_Data

##sale\_product data and external\_factors

DROP VIEW IF EXISTS Inventory\_Date;

-- Alternatively, if it's a table, use:

-- DROP TABLE IF EXISTS Inventory\_Date;

CREATE VIEW Inventory\_Date AS

SELECT

sp.Product\_ID,

sp.Sales\_Date,

sp.Inventory\_Quantity,

sp.Product\_Cost,

sp.Product\_Category,

sp.Promotions,

e.GDP,

e.Inflation\_Rate,

e.Seasonal\_Factor

FROM

Sales\_Product\_Data sp

JOIN

external\_factors e

ON

sp.Sales\_Date = e.New\_Sales\_Date;

Select \* from Inventory\_Date

## Descriptive Analysis

## Basic Statistics

## Avergae Sales (Inventory Quantity \* Product Cost\*)

Select Product\_ID,

round(AVG(Inventory\_Quantity\*Product\_Cost),2)AS avg\_sales

From Inventory\_Date

Group by 1

Order by avg\_sales DESC;

##Median stock levels (i., 'Inventory Quantity')

Select Product\_ID,

round(AVG(Inventory\_Quantity),2)AS median\_stock

From (

Select Product\_ID,

Inventory\_Quantity,

ROW\_NUMBER () OVER(Partition by Product\_ID order by Inventory\_Quantity) As row\_num\_asc,

ROW\_NUMBER () OVER(Partition by Product\_ID order by Inventory\_Quantity DESC) As row\_num\_desc

from Inventory\_Date)

AS subquery

Where row\_num\_asc IN (row\_num\_desc, row\_num\_desc -1, row\_num\_desc +1)

Group by Product\_ID;

##Product performance metrics (total sales per product)

Select Product\_ID,

round(sum(Inventory\_Quantity\*Product\_Cost)) as total\_sales

From Inventory\_Date

Group by 1

Order by total\_sales desc;

##Identify high-demand products based on avaerga sales

WITH HighDemandProducts AS (

SELECT Product\_ID, AVG(Inventory\_Quantity) AS avg\_sales

FROM Inventory\_Date

GROUP BY Product\_ID

HAVING avg\_sales > (SELECT AVG(Inventory\_Quantity) \* 0.95 FROM sales\_data)

)

## Calculate stockout frequency from high-demand products

select s.Product\_ID,

Count(\*) as stockout\_frequency

From Inventory\_Date s

Where S.Product\_ID in (Select Product\_ID from HighDemandProducts)

AND s.Inventory\_Quantity=0

Group by s.Product\_ID;

## None of the High demand products has experienced stockout

## Alternative way to Calculate stockout frequency from high-demand products

WITH HighDemandProducts AS (

SELECT Product\_ID, AVG(Inventory\_Quantity) AS avg\_sales

FROM Inventory\_Date

GROUP BY Product\_ID

HAVING avg\_sales > (SELECT AVG(Inventory\_Quantity) \* 0.95 FROM sales\_data)

)

SELECT s.Product\_ID,

Count(\*) as stockout\_frequency

FROM Inventory\_Date s

JOIN HighDemandProducts hdp ON s.Product\_ID = hdp.Product\_ID

WHERE s.Inventory\_Quantity = 0

GROUP BY s.Product\_ID;

## Influence of external factors

## GDP is overall economic health. Lower GDP is an indicator of economic downturn

## High inflation rate can deter customers purchasing power.

## Influence of GDP

select \* from Inventory\_Date

Select Product\_ID, GDP,

Avg(Case when 'GDP'>=0 then Inventory\_Quantity else null end) As avg\_sales\_positive\_gdp,

Avg(Case when 'GDP'<0 then Inventory\_Quantity else null end) As avg\_sales\_non\_negative\_gdp

From Inventory\_Date

Group by 1,2

Having avg\_sales\_positive\_gdp Is NOT Null;

Select Product\_ID, Inflation\_Rate,

Avg(Case when 'Inflation\_Rate'>=0 then Inventory\_Quantity else null end) As avg\_sales\_positive\_gdp,

Avg(Case when 'Inflation\_Rate'<0 then Inventory\_Quantity else null end) As avg\_sales\_non\_negative\_gdp

From Inventory\_Date

Group by 1,2

Having avg\_sales\_positive\_gdp Is NOT Null;

##Inventory Optimisation aims to ensure the right stock is maintained to meet customers demand

## Determine the optimal reorder point

## Reorder Point, inventory level at which a new order should be placed.

## Reorder Point = Lead time demand + Safety Stock

## Lead time expected sales during the lead time.

## Lead time = Rolling average sales \* lead time.

## Safety stock, buffer stock to account for demand and supply.

## Safety stock = Z \* the root of lead time \* standard deviation of demand.

## Lead time between placing an order and receiving it

## A constant lead time of 7 days for all products

## We aim for a 95% service level

WITH InventoryCalculation AS (

SELECT

Product\_ID,

AVG(rolling\_avg\_sales) AS avg\_rolling\_sales,

AVG(rolling\_variance) AS avg\_rolling\_variance

FROM (

SELECT

Product\_ID,

AVG(daily\_sales) OVER (PARTITION BY Product\_ID ORDER BY Sales\_date ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) AS rolling\_avg\_sales,

AVG(squared\_diff) OVER (PARTITION BY Product\_ID ORDER BY Sales\_date ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) AS rolling\_variance

FROM (

SELECT

Product\_ID,

Sales\_Date,

Inventory\_Quantity \* Product\_Cost AS daily\_sales,

(Inventory\_Quantity \* Product\_Cost - AVG(Inventory\_Quantity \* Product\_Cost) OVER (PARTITION BY Product\_ID ORDER BY Sales\_date ROWS BETWEEN 6 PRECEDING AND CURRENT ROW)) \*

(Inventory\_Quantity \* Product\_Cost - AVG(Inventory\_Quantity \* Product\_Cost) OVER (PARTITION BY Product\_ID ORDER BY Sales\_date ROWS BETWEEN 6 PRECEDING AND CURRENT ROW)) AS squared\_diff

FROM Inventory\_Date

) subquery

) subquery2

GROUP BY Product\_ID

)

SELECT

Product\_ID,

avg\_rolling\_sales,

avg\_rolling\_variance,

avg\_rolling\_sales \* 7 AS lead\_time\_demand,

1.645 \* (avg\_rolling\_variance \* 7) AS safety\_stock,

(avg\_rolling\_sales \* 7) + (1.645 \* (avg\_rolling\_variance \* 7)) AS reorder\_point

FROM InventoryCalculation;

## Safety stock of 0 means they have to start re-ordering the products

## Create Optimisation Table

Create table inventory\_optimization (

Product\_ID int,

reorder\_point double

);

## Step 2: Create the stored Procedure to recalculate reorder point

DELIMITER //

CREATE PROCEDURE RecalculateReorderPoint(IN productID INT)

BEGIN

DECLARE avg\_rolling\_sales DOUBLE;

DECLARE avg\_rolling\_variance DOUBLE;

DECLARE lead\_time\_demand DOUBLE;

DECLARE safety\_stock DOUBLE;

DECLARE reorder\_point DOUBLE;

SELECT

AVG(rolling\_avg\_sales),

AVG(rolling\_variance)

INTO

avg\_rolling\_sales,

avg\_rolling\_variance

FROM (

SELECT

Product\_ID,

AVG(daily\_sales) OVER (PARTITION BY Product\_ID ORDER BY Sales\_date ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) AS rolling\_avg\_sales,

AVG(squared\_diff) OVER (PARTITION BY Product\_ID ORDER BY Sales\_date ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) AS rolling\_variance

FROM (

SELECT

Product\_ID,

Sales\_Date,

Inventory\_Quantity \* Product\_Cost AS daily\_sales,

(Inventory\_Quantity \* Product\_Cost - AVG(Inventory\_Quantity \* Product\_Cost) OVER (PARTITION BY Product\_ID ORDER BY Sales\_date ROWS BETWEEN 6 PRECEDING AND CURRENT ROW)) \*

(Inventory\_Quantity \* Product\_Cost - AVG(Inventory\_Quantity \* Product\_Cost) OVER (PARTITION BY Product\_ID ORDER BY Sales\_date ROWS BETWEEN 6 PRECEDING AND CURRENT ROW)) AS squared\_diff

FROM Inventory\_Date

WHERE Product\_ID = productID

) AS inner\_derived

) AS outer\_derived;

SET lead\_time\_demand = avg\_rolling\_sales \* 7;

SET safety\_stock = 1.645 \* SQRT(avg\_rolling\_variance \* 7);

SET reorder\_point = lead\_time\_demand + safety\_stock;

INSERT INTO inventory\_optimization (Product\_ID, reorder\_point)

VALUES (productID, reorder\_point)

ON DUPLICATE KEY UPDATE reorder\_point = reorder\_point;

END //

DELIMITER ;

## Step 3: make inventory\_data a permanent table

create table Inventory\_table as select \* from inventory\_Date

## Step 4: create the trigger

DELIMITER //

DROP TRIGGER IF EXISTS AfterInsertUnifiedTable;

CREATE TRIGGER AfterInsertUnifiedTable

AFTER INSERT ON Inventory\_table

FOR EACH ROW

BEGIN

CALL RecalculateReorderPoint(NEW,Product\_ID);

END//

DELIMITER ;

## Analysing overstocking and understocking products

## Overstocking refers to a situation where a company holds more inventory than is necessary to meet customer demand

## Understocking occurs when a company holds insufficient inventory to meet customer demand.

WITH RollingSales AS (

SELECT

Product\_ID,

Sales\_Date,

AVG(Inventory\_Quantity \* Product\_Cost) OVER (PARTITION BY Product\_ID ORDER BY Sales\_Date ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) AS rolling\_avg\_sales

FROM

inventory\_table

),

## Calculate the number of days a product was out of stock

StockoutDays AS (

SELECT

Product\_ID,

COUNT(\*) AS stockout\_days

FROM

inventory\_table

WHERE

Inventory\_Quantity = 0

GROUP BY

Product\_ID

)

## Join the above CTEs with the main table to get the results

SELECT

f.Product\_ID,

AVG(f.Inventory\_Quantity \* f.Product\_Cost) AS avg\_inventory\_value,

AVG(rs.rolling\_avg\_sales) AS avg\_rolling\_sales,

COALESCE(sd.stockout\_days, 0) AS stockout\_days

FROM

inventory\_table f

JOIN

RollingSales rs ON f.Product\_ID = rs.Product\_ID AND f.Sales\_Date = rs.Sales\_Date

LEFT JOIN

StockoutDays sd ON f.Product\_ID = sd.Product\_ID

GROUP BY

f.Product\_ID, sd.stockout\_days;

## Most of the products are constantly overstocked.

## No understocked products

## Monitor and Adjust Procedure

## Monitor inventory levels

DELIMITER //

CREATE PROCEDURE MonitorInventorylevels()

begin

Select Product\_ID, Avg(Inventory\_Quantity) as AvgInventory

From Inventory\_table

group by Product\_ID

order by AvgInventory desc;

END//

Delimiter ;

## Monitor Sales Trends

DELIMITER //

CREATE PROCEDURE MonitorSalesTrends()

begin

Select Product\_ID, Sales\_Date,

AVG(Inventory\_Quantity \* Product\_Cost) OVER (PARTITION BY Product\_ID ORDER BY Sales\_Date ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) AS rolling\_avg\_sales

From Inventory\_table

order by Product\_ID, Sales\_Date;

END//

Delimiter ;

## Monitor Stockout Frequencies

DELIMITER //

CREATE PROCEDURE MonitorStockouts()

begin

Select Product\_ID, Count(\*) as StockoutDays

From Inventory\_table

where Inventory\_Quantity = 0

group by Product\_ID

order by StockoutDays desc;

END//

Delimiter ;

## Feedback Loop

## Feedback Loop Establishment:

## Feedback Portal: Develop an online platform for stakeholders to easily submit feedback on inventory performance and challenges.

## Review Meetings: Organize periodic sessions to discuss inventory system performance and gather direct insights.

## System Monitoring: Use established SQL procedures to track system metrics, with deviations from expectations flagged for review.

## Refinement Based on Feedback:

## Feedback Analysis: Regularly compile and scrutinize feedback to identify recurring themes or pressing issues.

## Action Implementation: Prioritize and act on the feedback to adjust reorder points, safety stock levels, or overall processes.

## Change Communication: Inform stakeholders about changes, underscoring the value of their feedback and ensuring transparency.

## Insights

## Inventory Discrepancies

## The initial stages of the analysis revealed significant discrepancies in inventory levels, with instances of overstocking. These inconsistencies were contributing to customer capital inefficiencies and dissatisfaction.

## Sales Trends and External Influences

## The analysis indicated that sales trends were notably influenced by various external factors. Recognizing these patterns presents an opportunity to forecast demand more accurately.

## Suboptimal Inventory Levels

## Through the inventory optimization analysis, it was evident that the existing inventory levels were not optimized for current sales trends. Products were identified that had either close excess inventory.

## Recommendations

## 1. Implement Dynamic Inventory Management: The company should transition from a static to a dynamic management system, adjusting inventory levels based on real-time sales trends, seasonality, and external factors.

## 2. Optimize Reorder Points and Safety Stocks: Use the reorder points and safety stocks calculated during the analysis to minimize stockouts and reduce excess inventory. Regularly review these metrics to ensure they align with current market conditions.

## 3. Enhance Pricing Strategies: Conduct a thorough review of product pricing strategies, especially for products identified as unprofitable. Consider factors such as competitor pricing, market demand, and product acquisition costs.

## 4. Reduce Overstock: Identify products that are consistently overstocked and take steps to reduce their inventory levels. This could include promotional sales, discounts, or even discontinuing products with low sales performance.

## 5. Establish a Feedback Loop: Develop a systematic approach to collect and analyze feedback from various stakeholders. Use this feedback for continuous improvement and alignment with business objectives.

## 6. Regular Monitoring and Adjustments: Adopt a proactive approach to inventory management by regularly monitoring key metrics and making necessary adjustments to inventory levels, order quantities, and safety stocks.