```
-- 📁 View all available databases
SHOW DATABASES;

-- 📂 Switch to the project-specific database
USE retail_db;

-- 📄 Show all tables in the current database
SHOW TABLES;

-- 🔍 Preview all data from the retail_data table
SELECT * FROM retail_data;

-- 📦 Total Inventory per Store
SELECT
    Store_ID,
    ROUND(SUM(Inventory_Level), 2) AS Total_Inventory
FROM
    retail_data
GROUP BY
    Store_ID
ORDER BY
    Total_Inventory DESC;

-- 📦 Total Inventory per Product
SELECT
    Product_ID,
    ROUND(SUM(Inventory_Level), 2) AS Total_Inventory
FROM
    retail_data
GROUP BY
    Product_ID
ORDER BY
    Total_Inventory DESC;

-- 📦 Inventory per Category and Region
SELECT
    Region,
    Category,
    ROUND(SUM(Inventory_Level), 2) AS Total_Inventory
FROM
    retail_data
GROUP BY
    Region, Category
ORDER BY
    Region, Total_Inventory DESC;

-- 🧾 Inventory per Product in Each Store
SELECT
```

```
    Store_ID,
    Product_ID,
    ROUND(SUM(Inventory_Level), 2) AS Total_Inventory
FROM
    retail_data
GROUP BY
    Store_ID, Product_ID
ORDER BY
    Store_ID, Total_Inventory DESC;

-- 🧾 Inventory per Category in Each Store
SELECT
    Store_ID,
    Category,
    ROUND(SUM(Inventory_Level), 2) AS Total_Inventory
FROM
    retail_data
GROUP BY
    Store_ID, Category
ORDER BY
    Store_ID, Total_Inventory DESC;



-- 📌 Reorder Point Estimation Using Moving Average and Lead Time

WITH demand_window AS (
    -- Step 1: Calculate 5-day moving average of units sold (past 5 days only)
    SELECT
        STR_TO_DATE(Date, '%Y-%m-%d') AS date_converted,
        Store_ID,
        Product_ID,
        Category,
        Inventory_Level,
        Units_Sold,
        ROUND(
            AVG(Units_Sold) OVER (
                PARTITION BY Store_ID, Product_ID
                ORDER BY STR_TO_DATE(Date, '%Y-%m-%d')
                ROWS BETWEEN 5 PRECEDING AND 1 PRECEDING
            ), 2
        ) AS avg_daily_demand
    FROM retail_data
),

add_lead_time AS (
    -- Step 2: Assign lead time based on product category
    SELECT *,
        CASE
            WHEN Category IN ('Toys', 'Clothing', 'Groceries') THEN 3
            WHEN Category IN ('Furniture', 'Electronics') THEN 5
            ELSE 4
        END AS lead_time
    FROM demand_window
),

reorder_calc AS (
    -- Step 3: Calculate Reorder Point = avg_daily_demand * lead_time
    SELECT
        date_converted AS Date,
        Store_ID,
        Product_ID,
        Category,
        Inventory_Level,
        avg_daily_demand,
        lead_time,
        ROUND(avg_daily_demand * lead_time, 2) AS reorder_point
    FROM add_lead_time
    WHERE avg_daily_demand IS NOT NULL
)

-- Step 4: Final result with Reorder Point and Low Inventory Risk flag
SELECT
    Date,
    Store_ID,
    Product_ID,
    Category,
    Inventory_Level,
    avg_daily_demand,
    lead_time,
    reorder_point,
    CASE
        WHEN Inventory_Level < reorder_point THEN 'Yes'
        ELSE 'No'
    END AS Low_Inventory_Risk
FROM reorder_calc
ORDER BY Date
LIMIT 1000;


-- 📊 Weekly Inventory Turnover for Each Product per Store
SELECT
    YEARWEEK(STR_TO_DATE(Date, '%Y-%m-%d'), 1) AS Week,
    Product_ID,
    Store_ID,
    ROUND(SUM(Units_Sold) / NULLIF(AVG(Inventory_Level), 0), 2) AS weekly_turnover
```

```
FROM retail_data
GROUP BY Week, Product_ID, Store_ID
ORDER BY weekly_turnover DESC;


-- 🧮 Quartile Classification of Weekly Inventory Turnover
WITH weekly_turnover_data AS (
    SELECT
        YEARWEEK(STR_TO_DATE(Date, '%Y-%m-%d'), 1) AS Week,
        Product_ID,
        Store_ID,
        ROUND(SUM(Units_Sold) / NULLIF(AVG(Inventory_Level), 0), 2) AS turnover
    FROM retail_data
    GROUP BY Week, Product_ID, Store_ID
),

ranked_turnover AS (
    -- Step 2: Divide into quartiles using NTILE
    SELECT
        *,
        NTILE(4) OVER (ORDER BY turnover) AS quartile
    FROM weekly_turnover_data
)

-- Step 3: Show quartile summary with min, max, and average turnover
SELECT
    quartile,
    ROUND(AVG(turnover), 2) AS avg_turnover,
    MIN(turnover) AS min_turnover,
    MAX(turnover) AS max_turnover
FROM ranked_turnover
GROUP BY quartile
ORDER BY quartile;


-- 📌 Weekly Turnover Categorization into Slow/Medium/Fast-Moving Products

-- Step 1: Calculate weekly turnover per product-store-category
WITH weekly_turnover_data AS (
    SELECT
        CONCAT(YEAR(STR_TO_DATE(Date, '%Y-%m-%d')), '-W',
LPAD(WEEK(STR_TO_DATE(Date, '%Y-%m-%d'), 1), 2, '0')) AS Week,
        Product_ID,
        Store_ID,
        Category,
        ROUND(SUM(Units_Sold) / NULLIF(AVG(Inventory_Level), 0), 2) AS turnover
    FROM retail_data
    GROUP BY Week, Product_ID, Store_ID, Category
```

```
),

-- Step 2: Label each row by turnover performance category
labeled_turnover AS (
    SELECT
        Week,
        Product_ID,
        Store_ID,
        Category,
        turnover,
        CASE
            WHEN turnover <= 4.30 THEN 'Slow-Moving'
            WHEN turnover <= 4.95 THEN 'Medium-Moving'
            ELSE 'Fast-Moving'
        END AS Turnover_Category
    FROM weekly_turnover_data
)

-- Step 3: Final result
SELECT *
FROM labeled_turnover
ORDER BY turnover DESC
LIMIT 1000;

-- % of rows where Units_Sold > Inventory_Level
SELECT
    ROUND(100.0 * SUM(CASE WHEN Units_Sold > Inventory_Level THEN 1 ELSE 0 END)
/ COUNT(*), 2) AS stockout_rate_percent
FROM retail_data;

SELECT
    Product_ID,
    ROUND(
        100.0 * SUM(CASE WHEN Units_Sold > Inventory_Level THEN 1 ELSE 0 END) /
COUNT(*),
        2
    ) AS stockout_rate_percent
FROM retail_data
GROUP BY Product_ID
ORDER BY stockout_rate_percent DESC
LIMIT 20; -- top 20 most affected products

SELECT
    Store_ID,
    ROUND(
        100.0 * SUM(CASE WHEN Units_Sold > Inventory_Level THEN 1 ELSE 0 END) /
COUNT(*),
        2
    ) AS stockout_rate_percent
FROM retail_data
GROUP BY Store_ID
ORDER BY stockout_rate_percent DESC;

SELECT
    Category,
    ROUND(
        100.0 * SUM(CASE WHEN Units_Sold > Inventory_Level THEN 1 ELSE 0 END) /
COUNT(*),
        2
    ) AS stockout_rate_percent
FROM retail_data
GROUP BY Category
ORDER BY stockout_rate_percent DESC;

SELECT
    Region,
    ROUND(
        100.0 * SUM(CASE WHEN Units_Sold > Inventory_Level THEN 1 ELSE 0 END) /
COUNT(*),
        2
    ) AS stockout_rate_percent
FROM retail_data
GROUP BY Region
ORDER BY stockout_rate_percent DESC;


SELECT
    Product_ID,
    Store_ID,
    ROUND(
        100.0 * SUM(CASE WHEN Units_Sold > Inventory_Level THEN 1 ELSE 0 END) /
COUNT(*),
        2
    ) AS stockout_rate_percent
FROM retail_data
GROUP BY Product_ID, Store_ID
HAVING stockout_rate_percent > 0
ORDER BY stockout_rate_percent DESC
LIMIT 20;

SELECT
    Product_ID,
    ROUND(AVG(Inventory_Level), 2) AS avg_inventory_level
FROM retail_data
GROUP BY Product_ID
ORDER BY avg_inventory_level DESC
```

```
LIMIT 20;


SELECT
   Store_ID,
   ROUND(AVG(Inventory_Level), 2) AS avg_inventory_level
FROM retail_data
GROUP BY Store_ID
ORDER BY avg_inventory_level DESC;


SELECT
   Category,
   ROUND(AVG(Inventory_Level), 2) AS avg_inventory_level
FROM retail_data
GROUP BY Category
ORDER BY avg_inventory_level DESC;

SELECT
   Region,
   ROUND(AVG(Inventory_Level), 2) AS avg_inventory_level
FROM retail_data
GROUP BY Region
ORDER BY avg_inventory_level DESC;


SELECT
   Product_ID,
   Store_ID,
   ROUND(AVG(Inventory_Level), 2) AS avg_inventory_level
FROM retail_data
GROUP BY Product_ID, Store_ID
ORDER BY avg_inventory_level DESC
LIMIT 20;
```