TUNIS BUSINESS SCHOOL
UNIVERSITY OF TUNIS

Business Intelligence Mini-project

Maven Toys Database Management and Analysis

Participants:

Chaima Chammaa
*ID: ****6736*
Email: chaimachammaa9@gmail.com

Elaa Marco
*ID: ****7089*
Email: Ella.marco.tn@gmail.com

IT-300 Business Intelligence and Database Management Systems
Pr. Manel Abdelkader & Pr. Ameni Azzouz

GitHub Repository: Maven Toys BI Project

Contents

1 Introduction

This Business Intelligence (BI) project focuses on analyzing a comprehensive dataset containing information about sales, inventory, products, and store locations. The goal is to uncover meaningful insights into product performance, sales trends, and inventory management to support strategic decision-making.

Using PostgreSQL as the backend database, we designed a robust data warehouse to store and organize the data efficiently. The ETL (Extract, Transform, Load) process ensured that the data was cleaned, transformed, and loaded into the warehouse for analysis. To visualize the data, we used Power BI to create an intuitive and interactive dashboard.

This dashboard presents key metrics and KPIs, enabling the company's management to:
- Identify high-performing products and categories.
- Optimize inventory levels across stores.
- Evaluate sales performance by region and store.
- Make data-driven decisions for future expansion, marketing strategies, and catalog optimization.

2 Implementation

2.1 Data gathering

Data Sources:
The Maven Toys dataset was extracted from Maven Analytics, a reliable source for business intelligence datasets. The dataset includes the following files:

Sales data: Information about sales transactions (sales.csv).
Inventory data: Details about stock levels across stores (inventory.csv).
Store data: Information about store locations and attributes (stores.csv).
Product data: Details about products, including categories and pricing (products.csv).

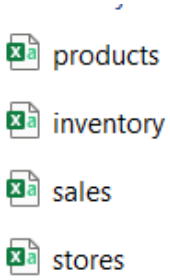

Figure 1: Dataset files overview

2.2 Data Preparation (ETL Process in PostgreSQL)

2.2.1 Data Loading:

Raw data was loaded into PostgreSQL using the COPY command:

```
-- Load data into stores table
COPY stores (Store_ID, Store_Name, Store_City, Store_Location, Store_Open_Date)
FROM 'C:/Users/Dell/Downloads/Maven+Toysrawdata/Maven Toys Data/stores.csv'
WITH CSV HEADER;

-- Load data into products table
COPY products (Product_ID, Product_Name, Product_Category, Product_Cost, Product_Price)
FROM 'C:/Users/Dell/Downloads/Maven+Toysrawdata/Maven Toys Data/products.csv'
WITH CSV HEADER;

-- Load data into inventory table
COPY inventory (Store_ID, Product_ID, Stock_On_Hand)
FROM 'C:/Users/Dell/Downloads/Maven+Toysrawdata/Maven Toys Data/inventory.csv'
WITH CSV HEADER;

-- Load data into sales table
COPY sales (Sale_ID, Store_ID, Product_ID, Sale_Date, Quantity_Sold)
FROM 'C:/Users/Dell/Downloads/Maven+Toysrawdata/Maven Toys Data/sales.csv'
WITH CSV HEADER;
```

Figure 2: Data loading process in PostgreSQL

2.2.2 Data Cleaning and Transformation:

Data was cleaned and transformed directly within PostgreSQL using SQL queries:

```sql
-- Check for missing values in all tables
SELECT * FROM stores WHERE store_id IS NULL OR store_name IS NULL OR store_city IS NULL;

SELECT * FROM products WHERE product_id IS NULL OR product_name IS NULL OR product_category IS NULL;

SELECT * FROM inventory WHERE store_id IS NULL OR product_id IS NULL OR stock_on_hand IS NULL;

SELECT * FROM sales WHERE sale_id IS NULL OR store_id IS NULL OR product_id IS NULL OR sale_date IS NULL

-- Check for duplicate rows in the all tables
SELECT store_id, COUNT(*) FROM stores GROUP BY store_id HAVING COUNT(*) > 1;

SELECT product_id, COUNT(*) FROM products GROUP BY product_id HAVING COUNT(*) > 1;

SELECT store_id, product_id, COUNT(*) FROM inventory GROUP BY store_id, product_id HAVING COUNT(*) > 1;

SELECT sale_id, COUNT(*) FROM sales GROUP BY sale_id HAVING COUNT(*) > 1;

-- Check for invalid data types in the stores table (store_id should be an integer)
SELECT * FROM stores WHERE store_id::text !~ '^\d+$';

-- Check for invalid data types in the products table (product_id should be integer)
SELECT * FROM products WHERE product_id::text !~ '^\d+$';

-- Check for invalid data types in the inventory table (store_id and product_id should be integer)
SELECT * FROM inventory WHERE store_id::text !~ '^\d+$' OR product_id::text !~ '^\d+$';

-- Check for invalid data types in the sales table (store_id and product_id should be integer)
SELECT * FROM sales WHERE store_id::text !~ '^\d+$' OR product_id::text !~ '^\d+$';
```

Figure 3: Data cleaning and transformation process in PostgreSQL

2.3 Data Storage and Modeling

2.3.1 Snowflake Schema Design

The Snowflake Schema was chosen for its normalized structure, which reduces redundancy and improves data integrity. The schema includes:

- Fact Table: sales_fact (sale_id, product_id, store_id, date_id, quantity_sold, total_revenue)

```sql
-- Create Fact Table
CREATE TABLE sales_fact (
    sale_id INT PRIMARY KEY,
    product_id INT REFERENCES products_dim(product_id),
    store_id INT REFERENCES stores_dim(store_id),
    date_id INT REFERENCES time_dim(date_id),
    quantity_sold INT,
    total_revenue NUMERIC(10, 2));
```

Figure 4: Fact table structure in Snowflake schema

- Dimension Tables:

- products_dim (product_id, product_name, category_id)
- categories_dim (category_id, category_name)
- stores_dim (store_id, store_name, location_id)
- locations_dim (location_id, store_city, store_region)
- time_dim (date_id, sale_date, sale_month, sale_quarter, sale_year)

```sql
-- Create Dimension Tables
CREATE TABLE products_dim (
    product_id INT PRIMARY KEY,
    product_name VARCHAR(100),
    category_id INT REFERENCES categories_dim(category_id));

CREATE TABLE categories_dim (
    category_id INT PRIMARY KEY,
    category_name VARCHAR(50));

CREATE TABLE stores_dim (
    store_id INT PRIMARY KEY,
    store_name VARCHAR(100),
    location_id INT REFERENCES locations_dim(location_id));

CREATE TABLE locations_dim (
    location_id INT PRIMARY KEY,
    store_city VARCHAR(50),
    store_region VARCHAR(50));

CREATE TABLE time_dim (
    date_id INT PRIMARY KEY,
    sale_date DATE,
    sale_month INT,
    sale_quarter INT,
    sale_year INT);
```

Figure 5: Dimension tables structure in Snowflake schema

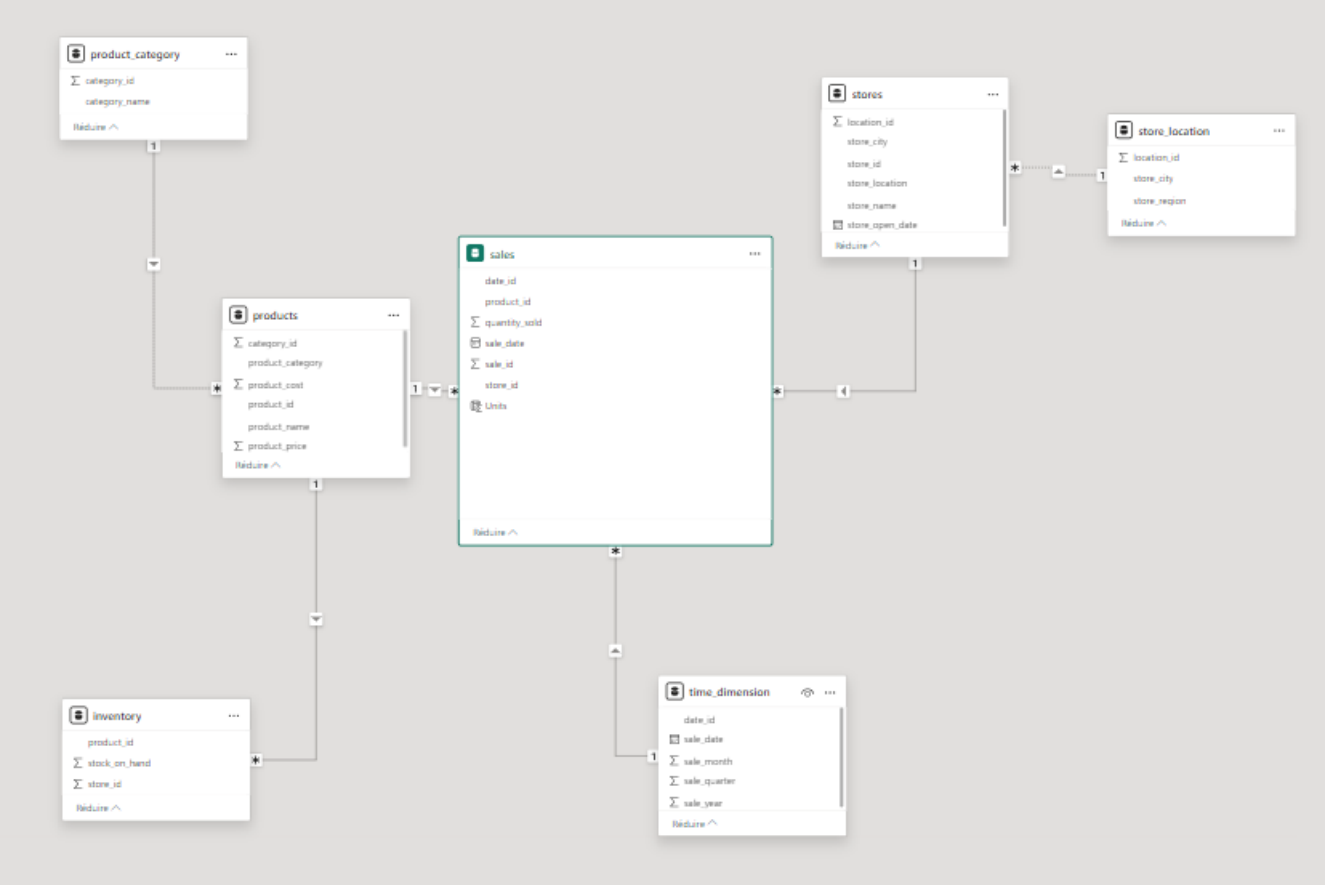

Figure 6: Snowflake schema

Also, aggregated views and data marts were created to simplify the analysis in Power BI:

```sql
-- | Create a sales data mart with joined tables
CREATE TABLE sales_mart AS
SELECT s.sale_id, s.store_id, s.product_id, s.sale_date, s.quantity_sold,
       st.store_name, st.store_city, p.product_name, p.product_category
FROM sales s
JOIN stores st ON s.store_id = st.store_id
JOIN products p ON s.product_id = p.product_id;

--  Create a materialized view for monthly sales by store
CREATE MATERIALIZED VIEW monthly_sales_by_store AS
SELECT DATE_TRUNC('month', sale_date) AS sale_month, store_id, SUM(quantity_sold) AS total_sales
FROM sales
GROUP BY sale_month, store_id;

--  Create a store performance data mart
CREATE TABLE store_performance AS
SELECT st.store_id, st.store_name, st.store_city,
       SUM(s.quantity_sold) AS total_sales,
       SUM(s.quantity_sold * p.product_price) AS total_revenue
FROM sales s
JOIN stores st ON s.store_id = st.store_id
JOIN products p ON s.product_id = p.product_id
GROUP BY st.store_id, st.store_name, st.store_city;
```

Figure 7: Aggregated views and data marts creation

2.3.2 ROLAP Queries for Analysis

ROLAP (Relational OLAP) was used to perform analytical queries directly on the relational database, ensuring efficient data processing and real-time insights. Below are the key ROLAP queries used:

```sql
-- ROLAP Queries
-- Total Sales by Store
SELECT store_name, SUM(quantity_sold) AS total_sales
FROM sales_mart
GROUP BY store_name;

-- Monthly Sales Trend
SELECT sale_month, SUM(total_sales) AS total_sales
FROM monthly_sales_by_store
GROUP BY sale_month;

-- Top-Selling Products
SELECT product_name, SUM(quantity_sold) AS total_sales
FROM sales_mart
GROUP BY product_name
ORDER BY total_sales DESC
LIMIT 10;

-- Revenue by Store
SELECT store_name, SUM(total_revenue) AS total_revenue
FROM store_performance
GROUP BY store_name;
```

Figure 8: ROLAP Queries

2.4 Data Analysis and Visualization

2.4.1 Data connection

Power BI provides a built-in PostgreSQL connector that allows direct connection to the database. The connector was configured with the necessary credentials (server name, database name, username, and password). The transformed and aggregated tables/views were imported into Power BI, and data refresh settings were configured to ensure the dashboard reflects the latest data from PostgreSQL. After importing, the data was validated in Power BI to ensure it matched the source data.

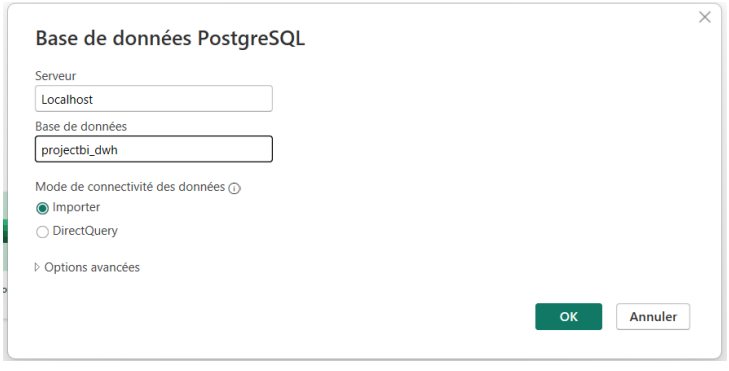

Figure 9: Database connection

2.4.2 Dashboard design

The Power BI dashboard was designed to provide actionable insights through Key Performance Indicators (KPIs) and interactive visualizations. The layout was organized into clear sections to highlight sales trends, revenue performance, inventory levels, and product performance. Each visualization was carefully chosen to ensure clarity and ease of interpretation, enabling users to make data-driven decisions efficiently. Below are the key components of the dashboard, supported by screenshots for reference.

Figure 10: Sales overview dashboard

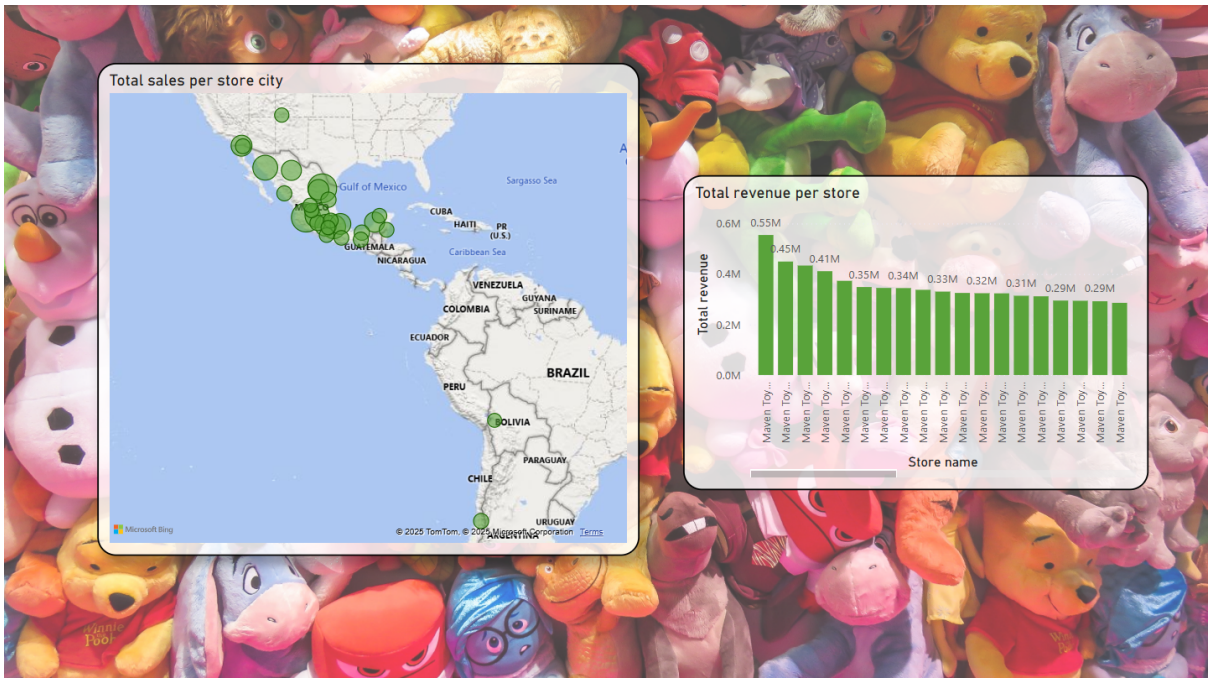

Figure 11: Store performance dashboard

We tried to add a touch of interactivity in our dashboard using slicers feature in Power BI to enhance user experience and enable dynamic filtering of the data.

For example:

- A Year slicer allows users to filter data by year (2022, 2023).
- A Store Name slicer allows users to filter data by specific stores
- A Product Category slicer allows users to filter data by category (Toys, Electronics, Art and Crafts).

2.4.3 Insights gained

Brief insights from each page:

- **Sales overview page**
  - Sales peaked during Q4 2022 (holiday season), and 2023 showed consistent growth, with a noticeable spike in July and August (likely due to summer promotions or back-to-school shopping).
- **Revenue analysis page**
  - The Toys category generated the highest revenue (35.26%), followed by Electronics (15.55%), indicating that these categories are the most profitable and should be prioritized in marketing and inventory planning.
- **Inventory management page**
  - Most stores have the largest stock on hand for Arts & Crafts, suggesting potential overstocking in this category. This highlights the need for better inventory optimization to reduce excess stock.
- **Store performance page**
  - Stores in Ciudad de Mexico and Guadalajara generated the highest revenue, while stores in smaller cities (Hermosillo, Guanajuato) underperformed, indicating a need for targeted strategies to improve sales in these regions.
- **Product performance page**
  - Products like Magic Sand and Kids Makeup Kit were top revenue generators, making them key drivers of profitability. These products should be prioritized in marketing and inventory planning.
- **Best selling products page**
  - The Top 10 Products accounted for a significant portion of total sales, with Toys and Electronics dominating the list. This suggests that focusing on these high-performing products can drive overall sales growth.

3 Conclusion

This Business Intelligence project successfully analyzed sales, revenue, inventory, and product performance data to provide actionable insights for the company. By leveraging PostgreSQL for data storage and transformation and Power BI for visualization, we created an interactive dashboard that highlights key trends and performance metrics.