A red and white logo with a bird

Description automatically generated

Wroclaw University of Technology

and

Telecommunications

**Data Warehouses Report 2**

Members:

Khalid Muzaffar(269553)

Mert Nuri Kodzhaaslan(260312)

**Task 1.1: Global Sales Information**

SELECT

SUM(so.SubTotal) AS SalesAmount,

SUM(so.TotalDue) AS Volume,

COUNT(\*) AS NumberOfOrders

FROM Sales.SalesOrderHeader AS so;

**Task 1.2: Sales Information by Year**

SELECT

YEAR(so.OrderDate) AS Year,

SUM(so.SubTotal) AS SalesAmount,

SUM(so.TotalDue) AS Volume,

COUNT(\*) AS NumberOfOrders

FROM Sales.SalesOrderHeader AS so

GROUP BY YEAR(so.OrderDate);

**Task 1.3: Top 5 Customers by Number of Orders**

SELECT TOP 5

c.CustomerID,

(p.FirstName + ' ' + p.LastName) AS FullName,

COUNT(\*) AS NumberOfOrders

FROM Sales.SalesOrderHeader AS so

JOIN Sales.Customer AS c ON so.CustomerID = c.CustomerID

JOIN Person.Person AS p ON c.PersonID = p.BusinessEntityID

GROUP BY c.CustomerID, p.FirstName, p.LastName

ORDER BY NumberOfOrders DESC;

**Task 1.4: Customers with Purchases Over $1500**

SELECT

c.CustomerID,

(p.FirstName + ' ' + p.LastName) AS FullName,

SUM(so.SubTotal) AS SalesAmount

FROM Sales.SalesOrderHeader AS so

JOIN Sales.Customer AS c ON so.CustomerID = c.CustomerID

JOIN Person.Person AS p ON c.PersonID = p.BusinessEntityID

GROUP BY c.CustomerID, p.FirstName, p.LastName

HAVING SUM(so.SubTotal) > 1500

ORDER BY SalesAmount DESC;

**Task 1.5: Product Categories Information**

SELECT

pc.ProductCategoryID AS CategoryID,

pc.Name AS CategoryName,

AVG(pr.ListPrice) AS AveragePrice,

SUM(sd.LineTotal) AS TotalSalesAmount,

SUM(sd.OrderQty) AS TotalVolume

FROM Production.Product AS pr

JOIN Production.ProductSubcategory AS psc ON pr.ProductSubcategoryID = psc.ProductSubcategoryID

JOIN Production.ProductCategory AS pc ON psc.ProductCategoryID = pc.ProductCategoryID

JOIN Sales.SalesOrderDetail AS sd ON pr.ProductID = sd.ProductID

GROUP BY pc.ProductCategoryID, pc.Name;

**Task 1.6: Subcategories with Higher Average Price than Categories**

WITH OverallAveragePrice AS (

    SELECT AVG(ListPrice) AS AveragePrice

    FROM Production.Product

)

SELECT

    psc.ProductSubcategoryID AS "SubcategoryID",

    psc.Name AS "Subcategory name",

    AVG(pr.ListPrice) AS "Average price",

    (SELECT AveragePrice FROM OverallAveragePrice) AS "Average price (over all categories)"

FROM Production.Product pr

JOIN Production.ProductSubcategory psc ON pr.ProductSubcategoryID = psc.ProductSubcategoryID

GROUP BY psc.ProductSubcategoryID, psc.Name

HAVING AVG(pr.ListPrice) > (SELECT AveragePrice FROM OverallAveragePrice);

**Task 1.7: Sales Territory Performance in May 2013**

WITH MonthlySales AS (

    SELECT

        st.TerritoryID,

        st.Name AS TerritoryName,

        MONTH(so.OrderDate) AS SaleMonth,

        YEAR(so.OrderDate) AS SaleYear,

        SUM(so.TotalDue) AS MonthlySales

    FROM Sales.SalesOrderHeader so

    JOIN Sales.SalesTerritory st ON so.TerritoryID = st.TerritoryID

    GROUP BY st.TerritoryID, st.Name, MONTH(so.OrderDate), YEAR(so.OrderDate)

),

AverageMonthlySales AS (

    SELECT

        TerritoryID,

        AVG(MonthlySales) AS AvgMonthlySales

    FROM MonthlySales

    GROUP BY TerritoryID

)

SELECT

    ms.TerritoryID AS "Sales territory ID",

    ms.TerritoryName AS "Sales territory name",

    ms.MonthlySales AS "Sales (May 2013)",

    ams.AvgMonthlySales AS "Average monthly sales (per territory)"

FROM MonthlySales ms

JOIN AverageMonthlySales ams ON ms.TerritoryID = ams.TerritoryID

WHERE ms.SaleMonth = 5 AND ms.SaleYear = 2013 AND ms.MonthlySales > ams.AvgMonthlySales;

**Task 1.8: List of Sales Territories with Average Number of Orders**

;WITH CustomerOrderCounts AS (

    SELECT

        CustomerID,

        COUNT(\*) AS TotalOrders

    FROM Sales.SalesOrderHeader

    GROUP BY CustomerID

    HAVING COUNT(\*) > 10

),

TerritoryOrderCounts AS (

    SELECT

        soh.TerritoryID,

        COUNT(\*) AS NumberOfOrders

    FROM Sales.SalesOrderHeader soh

    INNER JOIN CustomerOrderCounts coc ON soh.CustomerID = coc.CustomerID

    GROUP BY soh.TerritoryID

)

SELECT

    TerritoryID AS "TerritoryID",

    AVG(NumberOfOrders \* 1.0) AS "Average number of orders",

    FLOOR(AVG(NumberOfOrders \* 1.0)) AS "Average number of orders (INT)"

FROM TerritoryOrderCounts

GROUP BY TerritoryID;

**Task 1.9: Monthly Sales Amount by Territory in 2013 with Difference to Previous Month**

;WITH MonthlyTerritorySales AS (

SELECT

st.TerritoryID,

st.Name AS TerritoryName,

MONTH(so.OrderDate) AS [Month],

SUM(so.TotalDue) AS MonthlySalesAmount

FROM Sales.SalesOrderHeader so

JOIN Sales.SalesTerritory st ON so.TerritoryID = st.TerritoryID

WHERE YEAR(so.OrderDate) = 2013

GROUP BY st.TerritoryID, st.Name, MONTH(so.OrderDate)

),

SalesWithPreviousMonth AS (

SELECT

TerritoryID,

TerritoryName,

[Month],

MonthlySalesAmount,

LAG(MonthlySalesAmount, 1, 0) OVER (PARTITION BY TerritoryID ORDER BY [Month]) AS PrevMonthSalesAmount

FROM MonthlyTerritorySales

)

SELECT

TerritoryID,

TerritoryName,

MonthlySalesAmount,

MonthlySalesAmount - PrevMonthSalesAmount AS DiffToPrevMonth

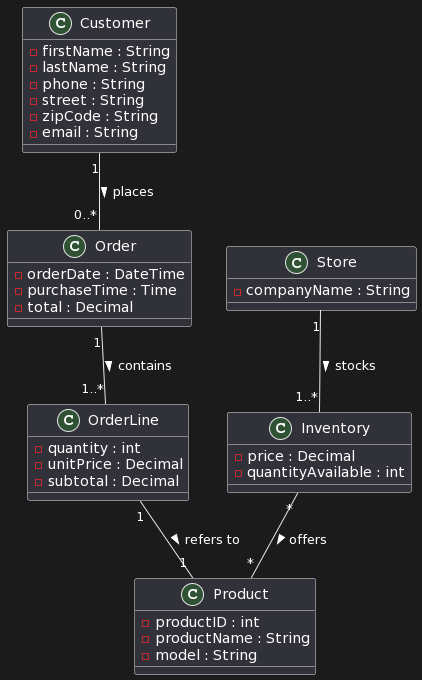
FROM SalesWithPreviousMonth

ORDER BY TerritoryID, [Month];

**Data Modelling**

1. R01 – A customer can repeatedly shop in the same store
   * This rule implies that there is a many-to-many relationship between the Customer and Store entities. The current model does not explicitly show this relationship. To correct the diagram, a new associative entity (e.g., CustomerStore) could be introduced to track the instances of customers shopping in stores or the Order entity can be used to imply this relationship as it captures the many-to-many interaction via the order placement relationship.
2. R02 – Any customer can shop in any store
   * This is already supported in the current model, but it must be ensured that there are no constraints that restrict a customer to a specific store or vice versa.
3. R03 – Each purchase is made by the customer in the store on a specific day and time
   * The Order entity should have attributes to capture the date and time of the purchase. This might require adding a purchaseTime attribute to the Order class.
4. R04 – A store must offer at least one product
   * This rule requires that there is at least a one-to-many relationship from Product to Store, which is not depicted in the current model. We should add a ProductList or Inventory entity that connects Store and Product with a one-to-many relationship from Store to ProductList.
5. R05 – The same product (type) can be offered by multiple stores
   * This is also not represented in the current model and ties into the need for a ProductList or Inventory entity to manage the many-to-many relationship between Product and Store.
6. R06 – Each store can individually propose the price and quantity of the offered product
   * This rule indicates that price and quantity should not be attributes of Product but of a relationship between Product and Store. This is where an Inventory entity or associative table would contain the price and quantityAvailable as attributes.
7. R07 – A customer must have a valid contact method
   * This new rule assumes that it's necessary to have contact information for a customer. We should add email or contactMethod as an attribute of Customer.
8. R08 – An order must contain at least one OrderLine
   * This is a new rule suggesting that an order is not valid unless it has one or more associated OrderLine items. This implies a one-to-many relationship from Order to OrderLine with a minimum cardinality of 1 on the OrderLine side.
9. R09 – Products must have a unique identifier
   * Products should have a productID as an attribute to uniquely identify them across different stores.
10. R10 – An order line must reference a single product
    * This rule enforces that each OrderLine is linked to exactly one Product. This is typically the case in an order system, and each OrderLine should have a foreign key that references the ProductID.

**2.**

****

**3.**

-- Create SEQUENCES for primary keys

CREATE SEQUENCE CustomerSeq START WITH 1 INCREMENT BY 1;

CREATE SEQUENCE StoreSeq START WITH 1 INCREMENT BY 1;

CREATE SEQUENCE ProductSeq START WITH 1 INCREMENT BY 1;

CREATE SEQUENCE OrderSeq START WITH 1 INCREMENT BY 1;

CREATE SEQUENCE OrderLineSeq START WITH 1 INCREMENT BY 1;

CREATE SEQUENCE InventorySeq START WITH 1 INCREMENT BY 1;

-- Create Customer Table

CREATE TABLE Customer (

CustomerID INT PRIMARY KEY DEFAULT (NEXT VALUE FOR CustomerSeq),

FirstName VARCHAR(50) NOT NULL,

LastName VARCHAR(50) NOT NULL,

Phone VARCHAR(15),

Street VARCHAR(100),

ZipCode VARCHAR(10),

Email VARCHAR(100) NOT NULL

);

-- Create Store Table

CREATE TABLE Store (

StoreID INT PRIMARY KEY DEFAULT (NEXT VALUE FOR StoreSeq),

CompanyName VARCHAR(100) NOT NULL,

RegistrationNumber VARCHAR(50) NOT NULL UNIQUE

);

-- Create Product Table

CREATE TABLE Product (

ProductID INT PRIMARY KEY DEFAULT (NEXT VALUE FOR ProductSeq),

ProductName VARCHAR(100) NOT NULL,

Model VARCHAR(50)

);

-- Create Order Table

CREATE TABLE [Order] (

OrderID INT PRIMARY KEY DEFAULT (NEXT VALUE FOR OrderSeq),

CustomerID INT NOT NULL,

StoreID INT NOT NULL,

OrderDate DATE NOT NULL,

PurchaseTime TIME NOT NULL,

Total DECIMAL(19, 4) NOT NULL CHECK (Total >= 0),

FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID),

FOREIGN KEY (StoreID) REFERENCES Store(StoreID)

);

-- Create OrderLine Table

CREATE TABLE OrderLine (

OrderLineID INT PRIMARY KEY DEFAULT (NEXT VALUE FOR OrderLineSeq),

OrderID INT NOT NULL,

ProductID INT NOT NULL,

Quantity INT NOT NULL CHECK (Quantity > 0),

UnitPrice DECIMAL(19, 4) NOT NULL CHECK (UnitPrice >= 0),

Subtotal AS (Quantity \* UnitPrice),

FOREIGN KEY (OrderID) REFERENCES [Order](OrderID),

FOREIGN KEY (ProductID) REFERENCES Product(ProductID)

);

-- Create Inventory Table

CREATE TABLE Inventory (

InventoryID INT PRIMARY KEY DEFAULT (NEXT VALUE FOR InventorySeq),

StoreID INT NOT NULL,

ProductID INT NOT NULL,

Price DECIMAL(19, 4) NOT NULL CHECK (Price > 0),

QuantityAvailable INT NOT NULL CHECK (QuantityAvailable >= 0),

UNIQUE (StoreID, ProductID),

FOREIGN KEY (StoreID) REFERENCES Store(StoreID),

FOREIGN KEY (ProductID) REFERENCES Product(ProductID)

);

**4.**

-- Correct Data Insertions

BEGIN TRANSACTION;

-- Insert Stores

INSERT INTO Store (CompanyName, RegistrationNumber) VALUES ('Tech Gadgets', 'RG001');

INSERT INTO Store (CompanyName, RegistrationNumber) VALUES ('Book World', 'RG002');

-- Insert Customers

INSERT INTO Customer (FirstName, LastName, Phone, Street, ZipCode, Email) VALUES ('Alice', 'Smith', '1234567890', '123 Apple St', '90001', 'alice@example.com');

INSERT INTO Customer (FirstName, LastName, Phone, Street, ZipCode, Email) VALUES ('Bob', 'Johnson', '0987654321', '456 Orange Ave', '90002', 'bob@example.com');

-- Insert Products

INSERT INTO Product (ProductName, Model) VALUES ('Laptop', 'L-123');

INSERT INTO Product (ProductName, Model) VALUES ('Smartphone', 'S-456');

-- Insert Inventory

INSERT INTO Inventory (StoreID, ProductID, Price, QuantityAvailable) VALUES (1, 1, 999.99, 10);

INSERT INTO Inventory (StoreID, ProductID, Price, QuantityAvailable) VALUES (2, 2, 199.99, 20);

-- Insert Orders

INSERT INTO [Order] (CustomerID, StoreID, OrderDate, PurchaseTime, Total) VALUES (1, 1, '2024-03-18', '10:00:00', 999.99);

INSERT INTO [Order] (CustomerID, StoreID, OrderDate, PurchaseTime, Total) VALUES (2, 2, '2024-03-18', '11:00:00', 399.98);

-- Insert OrderLines

INSERT INTO OrderLine (OrderID, ProductID, Quantity, UnitPrice) VALUES (1, 1, 1, 999.99);

INSERT INTO OrderLine (OrderID, ProductID, Quantity, UnitPrice) VALUES (2, 2, 2, 199.99);

COMMIT;

-- Incorrect Data Insertions

BEGIN TRANSACTION;

-- Try to insert a Store with no products (Should fail because of R04)

INSERT INTO Store (CompanyName, RegistrationNumber) VALUES ('Empty Store', 'RG003');

-- Try to insert an Order with negative total (Should fail because of constraint Total >= 0)

INSERT INTO [Order] (CustomerID, StoreID, OrderDate, PurchaseTime, Total) VALUES (1, 1, '2024-03-18', '12:00:00', -50.00);

-- Try to insert an OrderLine with negative quantity (Should fail because of constraint Quantity > 0)

INSERT INTO OrderLine (OrderID, ProductID, Quantity, UnitPrice) VALUES (1, 1, -1, 999.99);

ROLLBACK;