







Scenario: Creating a Data Warehouse with AdventureWorks

In this scenario, we are creating a data warehouse using the AdventureWorks database. AdventureWorks is a sample database provided by Microsoft that simulates an e-commerce system. The goal of the data warehouse is to consolidate and integrate data from multiple sources within the AdventureWorks database, enabling efficient reporting, analysis, and decision-making.

Definition of a Data Warehouse:

A data warehouse is a central repository that stores structured, historical data from various sources. It is designed to support business intelligence and reporting activities by providing a consolidated and consistent view of data for analysis and decision-making purposes. Data warehouses typically involve the extraction, transformation, and loading (ETL) of data from source systems into a dimensional model that optimizes querying and reporting performance.

Explanation of the Scenario:

In this scenario, we are using the AdventureWorks database as the source for our data warehouse. AdventureWorks contains tables representing different aspects of an e-commerce system, such as customers, orders, products, and sales. Our goal is to create a data warehouse that consolidates and integrates these tables to provide a unified view of the data.

To achieve this, we perform the following steps:

1. Data Extraction: We extract the relevant data from the AdventureWorks tables using SQL queries. This involves selecting the necessary columns and applying any necessary transformations or aggregations.

2. Data Transformation: We transform the extracted data to conform to the dimensional model of the data warehouse. This includes cleaning the data, resolving inconsistencies, and structuring it in a way that optimizes querying and reporting.

3. Data Loading: We load the transformed data into the appropriate tables of the data warehouse. These tables are designed using a dimensional model, typically consisting of fact tables and dimension tables. Fact tables contain the numerical and measurable data, while dimension tables provide the descriptive attributes.

4. Dimensional Modeling: We design and create the dimension and fact tables based on the business requirements. Dimension tables capture the descriptive attributes of the data, such as customer details, product information, and date dimensions. Fact tables store the numerical measures, such as sales quantities and amounts.

5. Reporting and Analysis: With the data warehouse in place, business users can perform various reporting and analysis tasks. They can generate reports, run ad-hoc queries, perform trend analysis, identify patterns, and gain insights to support decision-making processes. The data warehouse provides a unified and consistent view of the data, enabling efficient and accurate analysis.

By creating a data warehouse with AdventureWorks, we are able to consolidate and integrate data from various tables within the database, providing a solid foundation for reporting and analysis. The dimensional model and optimized structure of the data warehouse allow for efficient querying and reporting, empowering business users to gain valuable insights and make informed decisions based on the data.

-- 1. Retrieve the total number of orders.

SELECT COUNT(\*) AS TotalOrders

FROM Sales.SalesOrderHeader;

-- Explanation: This query provides the total number of orders in the database, which can be useful for understanding the overall order volume.

-- 2. Calculate the total revenue generated from sales.

SELECT SUM(TotalDue) AS TotalRevenue

FROM Sales.SalesOrderHeader;

-- Explanation: This query calculates the total revenue generated from sales by summing up the "TotalDue" column in the SalesOrderHeader table. It provides an overview of the overall revenue.

-- 3. Get the top 5 customers with the highest order count.

SELECT TOP 5 c.CustomerID, c.FirstName, c.LastName, COUNT(\*) AS OrderCount

FROM Sales.SalesOrderHeader o

JOIN Sales.Customer c ON o.CustomerID = c.CustomerID

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

ORDER BY OrderCount DESC;

-- Explanation: This query identifies the top 5 customers who have placed the highest number of orders. It helps identify the most active customers and their order count.

-- 4. Calculate the average order amount.

SELECT AVG(TotalDue) AS AverageOrderAmount

FROM Sales.SalesOrderHeader;

-- Explanation: This query calculates the average order amount by computing the average of the "TotalDue" column. It provides insight into the typical order value.

-- 5. Determine the total sales by product category.

SELECT p.ProductCategoryID, pc.Name AS CategoryName, SUM(od.LineTotal) AS TotalSales

FROM Sales.SalesOrderDetail od

JOIN Production.Product p ON od.ProductID = p.ProductID

JOIN Production.ProductCategory pc ON p.ProductCategoryID = pc.ProductCategoryID

GROUP BY p.ProductCategoryID, pc.Name

ORDER BY TotalSales DESC;

-- Explanation: This query calculates the total sales for each product category by summing up the line totals from the SalesOrderDetail table. It helps identify the highest-selling product categories.

-- 6. Identify the top-selling products by quantity.

SELECT TOP 10 p.ProductID, p.Name, SUM(od.OrderQty) AS TotalQuantitySold

FROM Sales.SalesOrderDetail od

JOIN Production.Product p ON od.ProductID = p.ProductID

GROUP BY p.ProductID, p.Name

ORDER BY TotalQuantitySold DESC;

-- Explanation: This query retrieves the top 10 best-selling products based on the total quantity sold. It helps identify the most popular products in terms of sales volume.

-- 7. Calculate the average profit margin by product.

SELECT p.ProductID, p.Name, AVG(ProfitMargin) AS AverageProfitMargin

FROM Production.ProductCostHistory pc

JOIN Production.Product p ON pc.ProductID = p.ProductID

GROUP BY p.ProductID, p.Name;

-- Explanation: This query calculates the average profit margin for each product based on historical cost data. It helps analyze the profitability of different products.

-- 8. Determine the number of orders per year.

SELECT YEAR(OrderDate) AS OrderYear, COUNT(\*) AS OrderCount

FROM Sales.SalesOrderHeader

GROUP BY YEAR(OrderDate)

ORDER BY OrderYear;

-- Explanation: This query provides the number of orders per year, allowing for analysis of order trends and patterns over time.

-- 9. Identify the top 5 markets with the highest revenue.

SELECT TOP 5 s.StoreID, s.Name AS Market, SUM(soh.TotalDue) AS TotalRevenue

FROM Sales.Store s

JOIN Sales.SalesOrderHeader soh ON s.BusinessEntityID = soh.SalesPersonID

GROUP BY s.StoreID, s.Name

ORDER BY TotalRevenue DESC;

-- Explanation: This query identifies the top 5 markets (stores) with the highest revenue by summing up the total revenue from sales orders. It helps identify the most profitable markets.

-- 10. Calculate the average order processing time.

SELECT AVG(DATEDIFF(DAY, OrderDate, ShipDate)) AS AverageProcessingTime

FROM Sales.SalesOrderHeader;

-- Explanation: This query calculates the average order processing time by computing the average number of days between the order date and the ship date. It helps evaluate order fulfillment efficiency.

-- 11. Determine the distribution of orders by status.

SELECT Status, COUNT(\*) AS OrderCount

FROM Sales.SalesOrderHeader

GROUP BY Status;

-- Explanation: This query provides the distribution of orders based on their status (e.g., pending, shipped, cancelled). It helps monitor order statuses and identify any potential issues.

-- 12. Get the top 10 customers with the highest total order amount.

SELECT TOP 10 c.CustomerID, c.FirstName, c.LastName, SUM(soh.TotalDue) AS TotalOrderAmount

FROM Sales.SalesOrderHeader soh

JOIN Sales.Customer c ON soh.CustomerID = c.CustomerID

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

ORDER BY TotalOrderAmount DESC;

-- Explanation: This query identifies the top 10 customers with the highest total order amount. It helps recognize the most valuable customers in terms of order value.