Practice Project: Introduction to Data Warehousing

Estimated time needed: 90 minutes

This comprehensive hands-on lab is designed to provide hands-on experience in designing and implementing a data warehouse. You have been hired as a data engineer for a consumer electronics retail company And the company wants you to create and implement a data warehouse to analyze its sales performance and inventory management.

What you will learn:

The lab offers various benefits, particularly for those seeking to enhance their data engineering and business intelligence skills.

Practical experience in data warehouse design: The lab provides hands-on experience in designing and implementing a star schema, which is crucial for any data warehousing project.

SQL Query writing skills: It enhances your ability to write complex SQL queries, including grouping sets, rollups, and cubes, essential for data analysis and reporting.

Data loading and transformation: The lab offers practice in data loading and transformation, an essential skill for managing data warehouses.

Real-world scenario applications: The scenario-based approach of the lab ensures that the skills acquired are relevant and applicable to real-world data warehousing and business intelligence projects.

Career advancement: These skills are in high demand in the fields of data engineering, business intelligence, and analytics, contributing significantly to professional growth and opportunities.

In a nutshell, this lab serves as a comprehensive guide for anyone aiming to strengthen their expertise in data warehousing and business intelligence, providing practical skills that are directly applicable in professional environments.

About SN Labs Cloud IDE

The Skills Network Labs Cloud IDE provides a hands-on environment for labs And utilizes Theia, an open-source Integrated Development Environment (IDE) platform that can run on a desktop or the cloud. To complete this lab, you will use the Cloud IDE based on Theia and PostgreSQL.

Single Session Exercise

Please be aware that sessions for this lab environment are not persistent. A new environment is created for you every time you connect to this lab. Any data you may have saved in an earlier session may get lost. To avoid losing your data, please plan to complete these labs in a single session.

Software used in the lab

In this lab, you will use PostgreSQL Database. PostgreSQL is a Relational Database Management System (RDBMS) designed to store, manipulate, and retrieve data efficiently.

Scenario

You are a data engineer hired by a consumer electronics retail company. The company sells various electronic products through its online and offline channels across major cities in the United States. They operate multiple stores and warehouses to manage their inventory and sales operations. The company wants to create a data warehouse to analyze its sales performance and inventory management and aim to generate reports, such as:

- Total sales revenue per year per city
- Total sales revenue per month per city
- Total sales revenue per quarter per city
- Total sales revenue per year per product category
- Total sales revenue per product category per city
- Total sales revenue per product category per store

Learning objectives

After completing this lab, you will be able to:

- Develop dimension and fact tables to organize and structure data effectively for analysis
- Employ SQL queries to create and load data into dimension and fact tables
- Create materialized views to optimize query performance

Note: Screenshots

Throughout this lab, you will be prompted to take screenshots and save them on your own device. You can use various free screengrabbing tools or your operating system’s shortcut keys (Alt + PrintScreen in Windows, Command + Shift + 5 on Mac, Shift + Ctrl + Show windows on Chromebook) to capture the required screenshots. The screenshots can be either jpg or png.

About the Dataset

The dataset you would be using in this assignment is not a real-life dataset. It was programmatically created for this project purpose.

Prerequisites

You need to use PostgreSQL Database to complete this lab.

You will start your project by designing a Star Schema warehouse by identifying the columns for the various dimensions and fact tables in the schema.

Exercise 1: Design a data warehouse

Sales ID	Product Type	Price Per Unit	Quantity Sold	City	Date
001	Electronics	$299.99	30	New York	2024-04-01
002	Apparel	$49.99	50	Los Angeles	2024-04-01
003	Furniture	$399.99	10	Chicago	2024-04-02
004	Electronics	$199.99	20	Houston	2024-04-02
005	Groceries	$2.99	100	Miami	2024-04-03

Task 1: Design the dimension table MyDimDateTask 1 - Design the dimension table MyDimDate

Write down the fields in the MyDimDate table in any text editor, one field per line. The company is looking at a granularity of day, which means they would like to have the ability to generate the report on a yearly, monthly, daily, and weekday basis.
Here is a partial list of fields to serve as an example:

dateid
month
monthname
…

Solution

Fields in MyDimDate table:

- dateid
- year
- month
- monthname
- day
- weekday
- weekdayname

The table will have a unique identifier (dateid) for each date entry. Other fields such as year, month, monthname, day, weekday, and weekdayname will provide detailed information about each date, allowing for flexible reporting options based on different time intervals.

Task 2: Design the dimension table MyDimProduct

Write down the fields in the MyDimProduct table in any text editor, one field per line.

Solution:

In this task, you will design the dimension table MyDimProduct to store product-related information.

Fields in MyDimProduct table:

- productid
- productname

The table will have a unique identifier (productid) for each product entry. The field productname will store the name or description of the product. This table will facilitate analysis and reporting based on different products sold.

Task 3: Design the dimension table MyDimCustomerSegment

Write down the fields in the MyDimCustomerSegment table in any text editor, one field per line.

Solution:

In this task, you will design the dimension table MyDimCustomerSegment to store customer segment-related information.

Fields in DimCustomerSegment table:

- segmentid
- segmentname

Task 4: Design the fact table MyFactSales

Write down the fields in the MyFactSales table in any text editor, one field per line.

Solution:

In this task, you will design the fact table MyFactSales to store sales-related information.

Fields in FactSales table:

- salesid
- productid
- quantitysold
- priceperunit
- segmentid
- dateid

The fact table FactSales will store information about each sales transaction, including the unique identifier (salesid), product identifier (productid), quantity sold (quantitysold), price per unit (priceperunit), customer segment identifier (segmentid) and date identifier (dateid). This table will serve as the central repository for sales data and enable analysis and reporting on various dimensions.

Exercise 2: Create schema for data warehouse on PostgreSQL

Open pgAdmin and create a database named **Practice**, then create the following tables.

Task 5: Create the dimension table MyDimDate

Create the MyDimDate table.

Take a screenshot of the SQL statement you used to create the table MyDimDate.

```
CREATE TABLE MyDimDate (
    dateid INT PRIMARY KEY,
    year INT,
    month INT,
    monthname VARCHAR(20),
    day INT,
    weekday INT,
    weekdayname VARCHAR(20)
);
```

You can execute this SQL statement in your preferred SQL database management tool, such as MySQL Workbench, pgAdmin, or SQL Server Management Studio, to create the DimDate table. Once created, you can verify the table structure and take a screenshot of it.

Task 6: Create the dimension table MyDimProduct

Create the MyDimProduct table.

```
CREATE TABLE MyDimProduct (
    productid INT PRIMARY KEY,
    productname VARCHAR(255)
);
```

This SQL statement will create a table named DimProduct with two columns: productid as the primary key and productname to store the name or description of the product. You can execute this SQL statement in your preferred SQL database management tool to create the DimProduct table.

Task 7: Create the dimension table MyDimCustomerSegment

```
CREATE TABLE MyDimCustomerSegment (
    segmentid INT PRIMARY KEY,
    segmentname VARCHAR(255)
);
```

This SQL statement will create a table named DimCustomerSegment with two columns: segmentid as the primary key and segmentname to store the name or description of the customer segment. You can execute this SQL statement in your preferred SQL database management tool to create the DimCustomerSegment table.

Task 8: Create the fact table MyFactSales

```
CREATE TABLE MyFactSales (
    salesid INT PRIMARY KEY,
    productid INT,
    quantitysold INT,
    priceperunit DECIMAL (10, 2),
    segmentid INT,
    dateid INT
);
```

This SQL statement will create a table named MyFactSales with the following columns:

- salesid: Primary key to uniquely identify each sales record.
- productid: Identifier for the product sold.
- quantitysold: Quantity of the product sold.
- priceperunit: Price per unit of the product sold.
- segmentid: Identifier for the customer segment.
- dateid: Identifier for the date of the transaction.

Exercise 3 - Load data into the data warehouse

In this exercise, you will load the data into the tables.

After the initial schema design, you were informed that data could not be collected in the format initially planned due to operational issues. This means that the previous tables (MyDimDate, MyDimProduct, MyDimCustomerSegment, MyFactSales) in the *Practice* database and their associated attributes are no longer applicable to the current design. The company has now provided data in CSV files according to the new design.

You will need to load the data provided by the company in CSV format. First, create a new database named **PracProj**. Then, create the tables DimDate, DimProduct, DimCustomerSegment, and FactSales as per the new schema. Follow the instructions in the lab to run the new scripts and load the data from the CSV files into the appropriate tables.

Task 9: Load data into the dimension table DimDate

Download the data from https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/-omGFpVSWBIZKFSCxUkBwg/DimDate.csv.

Load this data into DimDate table.

Take a screenshot of the first 5 rows in the table DimDate.

Name the screenshot `9-DimDate.jpg`. (Images can be saved with either the .jpg or .png extension.)

Solution:

Step 1: Open pgAdmin. Then create database **PracProj** and connect to PracProj database.

Step 2: Create the DimDate table

```
CREATE TABLE DimDate (
    Dateid INT PRIMARY KEY,
    date DATE NOT NULL,
    Year INT NOT NULL,
    Quarter INT NOT NULL,
    QuarterName VARCHAR(2) NOT NULL,
    Month INT NOT NULL,
    Monthname VARCHAR(255) NOT NULL,
    Day INT NOT NULL,
    Weekday INT NOT NULL,
    WeekdayName VARCHAR(255) NOT NULL
);
```

Step 3: Use the import tool in pgAdmin to load your CSV file into the table

- Navigate to the DimDate table in pgAdmin, right-click it, select “Import/Export”.

  Note: Ensure that you upload the files to this path: /var/lib/pgadmin/

- Choose “Import”, select your CSV file, and follow the prompts to import the data.

Step 4: Run this query to select the first 5 rows.

```
SELECT * FROM DimDate LIMIT 5;
```

After running the SELECT query, the first 5 rows of your DimDate table will be displayed in the query output panel.

Step 5: Take the screenshot.

Take a screenshot of this panel with the results. You can use the snipping tool on Windows, `Shift+Command+4` on Mac, or the equivalent on your operating system.

Task 10: Load data into the dimension table DimProduct

Download the data from https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/Y-76u4An3zb5R6HxxFPabA/DimProduct.csv.

Load this data into DimProduct table.
Take a screenshot of the first 5 rows in the table DimProduct.
Name the screenshot `10-DimProduct.jpg`. (Images can be saved with either the .jpg or .png extension.)

Solution:

Step 1: Create the DimProduct table

```
CREATE TABLE DimProduct (
    Productid INT PRIMARY KEY,
    Producttype VARCHAR(255) NOT NULL
);
```

Step 2: Use the import tool in pgAdmin to load your CSV file into the table

- Navigate to the DimProduct table in pgAdmin, right-click it, select "Import/Export".
- Choose "Import", select your CSV file, and follow the prompts to import the data.

Step 3: Run this query to select the first 5 rows

```
SELECT * FROM DimProduct LIMIT 5;
```

After running the SELECT query, the first 5 rows of your DimProduct table will be displayed in the query output panel.

Step 4: Take the screenshot.

Take a screenshot of this panel with the results. You can use the snipping tool on Windows, `Shift+Command+4` on Mac, or the equivalent on your operating system.

Task 11: Load data into the dimension table DimCustomerSegment

Download the data from https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/h_dnxb8yzQyVjeb8oYnm8A/DimCustomerSegment.csv.

Load this data into DimCustomerSegment table.

Take a screenshot of the first 5 rows in the table DimCustomerSegment.

Name the screenshot `11-DimCustomerSegment.jpg`. (Images can be saved with either the .jpg or .png extension.)

Solution:

Step 1: Create the DimCustomerSegment table

```
CREATE TABLE DimCustomerSegment (
    Segmentid INT PRIMARY KEY,
    City VARCHAR(255) NOT NULL
);
```

Step 2: Use the import tool in pgAdmin to load your CSV file into the table

- Navigate to the DimCustomerSegment table in pgAdmin, right-click it, select "Import/Export".
- Choose "Import", select your CSV file, and follow the prompts to import the data.

Step 3: Run this query to select the first 5 rows

```
SELECT * FROM DimCustomerSegment LIMIT 5;
```

After running the SELECT query, the first 5 rows of your DimCustomerSegment table will be displayed in the query output panel.

Step 4: Take the screenshot.

Take a screenshot of this panel with the results. You can use the snipping tool on Windows, `Shift+Command+4` on Mac, or the equivalent on your operating system.

Task 12: Load data into the fact table FactSales

Download the data from https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/a8kTjzvpdqzOp46ODatyAA/FactSales.csv

Load this data into FactSales table.

Take a screenshot of the first 5 rows in the table FactSales.

Name the screenshot `12-FactSales.jpg`. (Images can be saved with either the .jpg or .png extension.)

Solution:

Step 1: Create the FactSales table

```
CREATE TABLE FactSales (
    Salesid VARCHAR(255) PRIMARY KEY,
    Dateid INT NOT NULL,
    Productid INT NOT NULL,
    Segmentid INT NOT NULL,
    Price_PerUnit DECIMAL(10, 2) NOT NULL,
    QuantitySold INT NOT NULL,
    FOREIGN KEY (Dateid) REFERENCES DimDate(Dateid),
    FOREIGN KEY (Productid) REFERENCES DimProduct(Productid),
    FOREIGN KEY (Segmentid) REFERENCES DimCustomerSegment(Segmentid)
);
```

Step 2: Use the import tool in pgAdmin to load your CSV file into the table

- Navigate to the FactSales table in pgAdmin, right-click it, select "Import/Export".
- Choose "Import", select your CSV file, and follow the prompts to import the data.

Step 3: Run this query to select the first 5 rows

```
SELECT * FROM FactSales LIMIT 5;
```

After running the SELECT query, the first 5 rows of your FactSales table will be displayed in the query output panel.

Step 4: Take the screenshot.

Take a screenshot of this panel with the results. You can use the snipping tool on Windows, `Shift+Command+4` on Mac, or the equivalent on your operating system.

Exercise 4 - Write aggregation queries and create materialized view

In this exercise, you will query the data you have loaded in the previous exercise.

Task 13: Create a grouping sets query

Create a grouping sets query using the columns productid, producttype, total sales.
Take a screenshot of the SQL and the output rows.

Name the screenshot `13-groupingsets.jpg`. (Images can be saved with either the .jpg or .png extension.)

Solution:

```
SELECT
    p.Productid,
    p.Producttype,
    SUM(f.Price_PerUnit * f.QuantitySold) AS TotalSales
```

```
FROM
    FactSales f
INNER JOIN
    DimProduct p ON f.Productid = p.Productid
GROUP BY GROUPING SETS (
    (p.Productid, p.Producttype),
    p.Productid,
    p.Producttype,
    ()
)
ORDER BY
    p.Productid,
    p.Producttype;
```

In this query:

- You're joining FactSales 'f' with DimProduct 'p' on their productid to correlate each sale with its product type.

- You're using GROUPING SETS to specify the different levels of aggregation:

– By both Productid and Producttype
– By Productid alone
– By Producttype alone
– And a grand total with (), which doesn't group by any column and hence returns the sum for all sales.

- You're calculating TotalSales by multiplying the Price_PerUnit by QuantitySold for each sale.

The ORDER BY clause ensures the results are ordered by productid and then by producttype.

Task 14: Create a rollup query

Create a rollup query using the columns year, city, productid, and total sales. Take a screenshot of the SQL and the output rows.

Name the screenshot `14-rollup.jpg`. (Images can be saved with either the .jpg or .png extension.)

Solution:

```
SELECT
    d.Year,
    cs.City,
    p.Productid,
    SUM(f.Price_PerUnit * f.QuantitySold) AS TotalSales
FROM
    FactSales f
JOIN
    DimDate d ON f.Dateid = d.Dateid
JOIN
    DimProduct p ON f.Productid = p.Productid
JOIN
    DimCustomerSegment cs ON f.Segmentid = cs.Segmentid
GROUP BY ROLLUP (d.Year, cs.City, p.Productid)
ORDER BY
    d.Year DESC,
    cs.City,
    p.Productid;
```

This query performs the following operations:

- Joins FactSales with DimDate on Dateid, DimProduct on Productid, and DimCustomerSegment on Segmentid.
- Selects the year from DimDate, the city from DimCustomerSegment, and the product ID from DimProduct.
- Calculates total sales by multiplying the price per unit by the quantity sold for each sales entry.
- Groups the results using the ROLLUP function to create a grouping set that includes all combinations of year, city, and productid, along with their respective subtotals and a grand total for all sales.

The ORDER BY clause ensures the results are first ordered by year in descending order, then by city and product ID.

Task 15: Create a cube query

Create a cube query using the columns year, city, productid, and average sales.
Take a screenshot of the SQL and the output rows.
Name the screenshot `15-cube.jpg`. (Images can be saved with either the .jpg or .png extension.)

Solution:

```
SELECT
    d.Year,
    cs.City,
    p.Productid,
    AVG(f.Price_PerUnit * f.QuantitySold) AS AverageSales
FROM
    FactSales f
INNER JOIN
    DimDate d ON f.Dateid = d.Dateid
INNER JOIN
    DimProduct p ON f.Productid = p.Productid
INNER JOIN
    DimCustomerSegment cs ON f.Segmentid = cs.Segmentid
GROUP BY CUBE (d.Year, cs.City, p.Productid);
```

In this query:

- The CUBE clause is used in the GROUP BY to create subtotals for all combinations of year, city, and productid in addition to the grand total across all groups.

- AVG(f.Price_PerUnit * f.QuantitySold) calculates the average sales, factoring in both the price per unit and the quantity sold.

- INNER JOIN is used to join the FactSales table with the dimension tables DimDate, DimProduct, and DimCustomerSegment.

Task 16: Create a materialized view

Create an materialized view named max_sales using the columns city, productid, producttype, and max sales.
Take a screenshot of the SQL.
Name the `screenshot 16-mv.jpg`. (Images can be saved with either the .jpg or .png extension.)

Solution:

```
CREATE MATERIALIZED VIEW max_sales AS
SELECT
    cs.City,
    p.Productid,
    p.Producttype,
    MAX(f.Price_PerUnit * f.QuantitySold) AS MaxSales
FROM
```

```
        FactSales f
    JOIN
        DimProduct p ON f.Productid = p.Productid
    JOIN
        DimCustomerSegment cs ON f.Segmentid = cs.Segmentid
    GROUP BY
        cs.City,
        p.Productid,
        p.Producttype
    WITH DATA;
```

This statement will create the materialized view and populate it with the current data from the joined tables. The WITH DATA clause tells PostgreSQL to fill the view with the query results immediately. If you wanted to create the view without filling it with data, you would use WITH NO DATA.

To update the materialized view with the latest data, you would use the REFRESH MATERIALIZED VIEW command:

```
REFRESH MATERIALIZED VIEW max_sales;
```