Northwind Database

I. Introduction

1. About the Northwind database

The Northwind database is a database developed by Microsoft for guiding their database products over many decades. It contains information about the sales activities of a fictional company named "Northwind Traders," specializing in importing and exporting specialty foods from around the world. Northwind serves as an ideal schema for small business ERP tutorials, encompassing details about customers, orders, inventory, purchasing, suppliers, shipping, employees, and accounting.

2. About Business Requirements

This report dives into SQL-based Northwind analytics, addressing sales business questions through SQL queries. We transformed data from OLTP to OLAP, identified Sales facts and dimensions to easily analyze Northwind's sales results.

- Sales analysis: Provides comprehensive reporting on sales figures to understand which products are being sold to customers, which products sell the most, where they are sold, and which products sell the least. The goal is to get a comprehensive view of the business.

These analytics are intended to provide actionable insights and enhance the decision-making process for Northwind in terms of product sales.

3. Business question

The business process helps us understand what business questions we are asking. Here we focus on the what information is available; i.e., the attributes and relations between them. In Northwinds Trading Company database, we might want to try and answer some of these questions during the business modeling process:

- Total revenue, total profit, total sales, total orders, total employees, total customers
- Which country has the highest revenue?
- Which product category has the highest sales?
- Which company has the highest total revenue?
- Which product has the highest sales?
- How does revenue change over the months of the year?
- Which products are sold the most, where they are sold?
- Which products have the most Revenue?
- Which employees have the most total sales?
- Classify customers based on order frequency
- Which customer bought the most products?

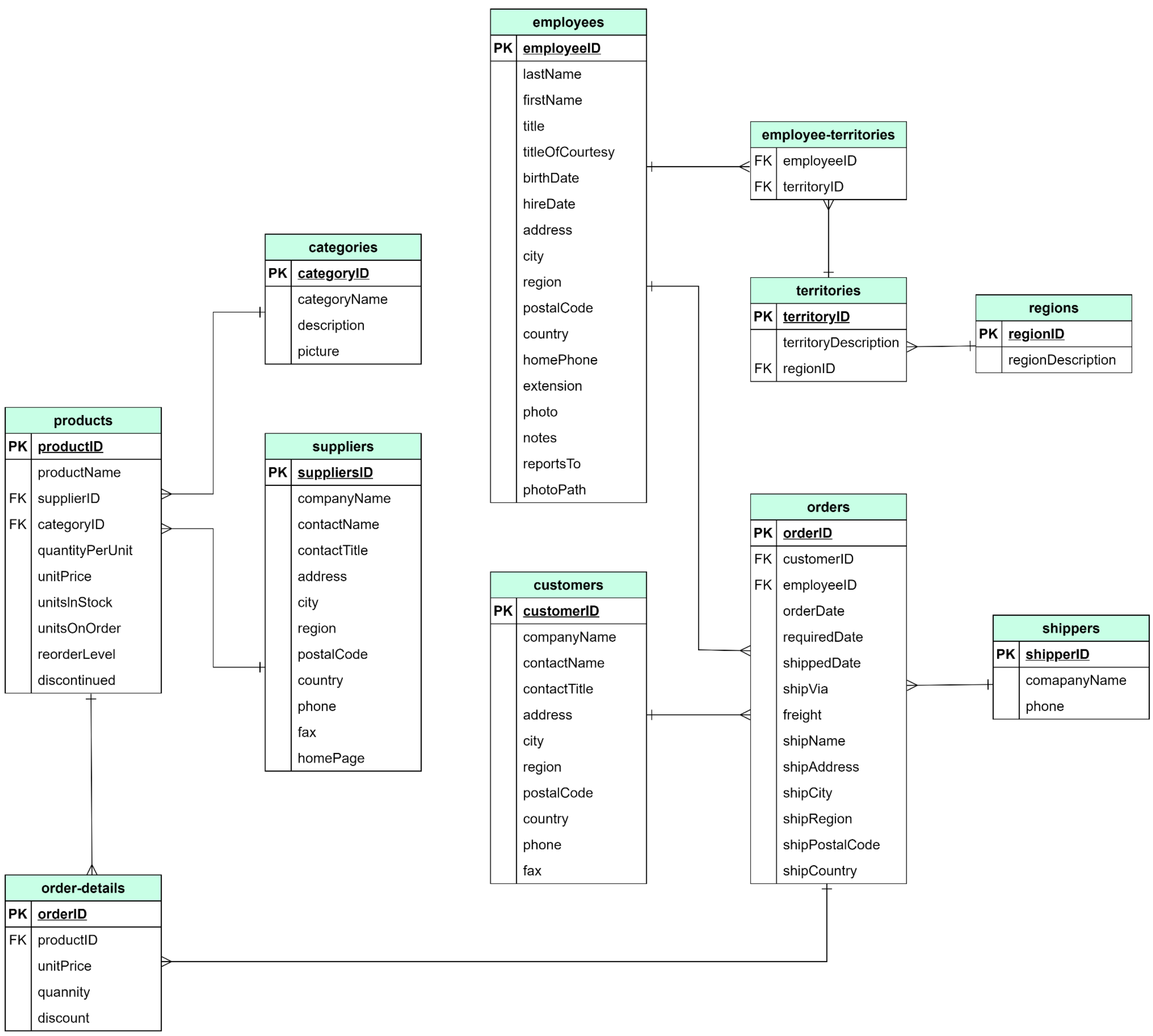
II. Database design

1. Design an entity-relationship diagram

Identifying required tables from ERD

- Customers - Individuals or entities purchasing goods from Northwind.
- Employees - Individuals working for Northwind.
- Employees territories - an entity linking employees with territories, indicating which territories are managed by specific employees.
- Categories - Categories of products that Northwind offers for sale.
- Orders - Transactions where orders are placed between customers and Northwind.
- Order Details - Detailed information about the items in orders placed by customers.
- Products - Products currently available for customers to purchase from Northwind.
- Regions - Entity that describes a geographic region.
- Shippers - Entities responsible for shipping orders from Northwind to customers.
- Suppliers - Entities supplying necessary items to Northwind.
- Territories - Another entity related to geographical territories, possibly used in conjunction with Employees Territories.

LINK: [TẠI ĐÂY]



2. Build a database based on the ERD

Employees

```
CREATE TABLE "Employees" (
    "EmployeeID" "int" IDENTITY (1, 1) NOT NULL ,
    "LastName" nvarchar (20) NOT NULL ,
    "FirstName" nvarchar (10) NOT NULL ,
    "Title" nvarchar (30) NULL ,
    "TitleOfCourtesy" nvarchar (25) NULL ,
    "BirthDate" "datetime" NULL ,
    "HireDate" "datetime" NULL ,
    "Address" nvarchar (60) NULL ,
    "City" nvarchar (15) NULL ,
    "Region" nvarchar (15) NULL ,
    "PostalCode" nvarchar (10) NULL ,
    "Country" nvarchar (15) NULL ,
    "HomePhone" nvarchar (24) NULL ,
    "Extension" nvarchar (4) NULL ,
    "Photo" "image" NULL ,
    "Notes" "ntext" NULL ,
    "ReportsTo" "int" NULL ,
    "PhotoPath" nvarchar (255) NULL ,
    CONSTRAINT "PK_Employees" PRIMARY KEY  CLUSTERED
    (
        "EmployeeID"
    ),
    CONSTRAINT "FK_Employees_Employees" FOREIGN KEY
    (
        "ReportsTo"
    ) REFERENCES "dbo"."Employees" (
        "EmployeeID"
    ),
    CONSTRAINT "CK_Birthdate" CHECK (BirthDate < getdate())
)

GO
 CREATE  INDEX "LastName" ON "dbo"."Employees"("LastName")
GO
 CREATE  INDEX "PostalCode" ON "dbo"."Employees"("PostalCode")
GO
```

Customers

```
CREATE TABLE "Customers" (
    "CustomerID" nchar (5) NOT NULL ,
    "CompanyName" nvarchar (40) NOT NULL ,
    "ContactName" nvarchar (30) NULL ,
    "ContactTitle" nvarchar (30) NULL ,
    "Address" nvarchar (60) NULL ,
    "City" nvarchar (15) NULL ,
    "Region" nvarchar (15) NULL ,
    "PostalCode" nvarchar (10) NULL ,
    "Country" nvarchar (15) NULL ,
    "Phone" nvarchar (24) NULL ,
    "Fax" nvarchar (24) NULL ,
    CONSTRAINT "PK_Customers" PRIMARY KEY  CLUSTERED
    (
        "CustomerID"
    )
)
GO
 CREATE  INDEX "City" ON "dbo"."Customers"("City")
GO
 CREATE  INDEX "CompanyName" ON "dbo"."Customers"("CompanyName")
GO
 CREATE  INDEX "PostalCode" ON "dbo"."Customers"("PostalCode")
GO
 CREATE  INDEX "Region" ON "dbo"."Customers"("Region")
GO
```

Categories

```
CREATE TABLE "Categories" (
    "CategoryID" "int" IDENTITY (1, 1) NOT NULL ,
    "CategoryName" nvarchar (15) NOT NULL ,
    "Description" "ntext" NULL ,
    "Picture" "image" NULL ,
    CONSTRAINT "PK_Categories" PRIMARY KEY  CLUSTERED
    (
        "CategoryID"
    )
)
GO
 CREATE  INDEX "CategoryName" ON "dbo"."Categories"("CategoryName")
GO
```

Shippers

```
CREATE TABLE "Shippers" (
    "ShipperID" "int" IDENTITY (1, 1) NOT NULL ,
    "CompanyName" nvarchar (40) NOT NULL ,
    "Phone" nvarchar (24) NULL ,
    CONSTRAINT "PK_Shippers" PRIMARY KEY  CLUSTERED
    (
        "ShipperID"
    )
)
GO
```

Suppliers

```
CREATE TABLE "Suppliers" (
    "SupplierID" "int" IDENTITY (1, 1) NOT NULL ,
    "CompanyName" nvarchar (40) NOT NULL ,
    "ContactName" nvarchar (30) NULL ,
    "ContactTitle" nvarchar (30) NULL ,
    "Address" nvarchar (60) NULL ,
    "City" nvarchar (15) NULL ,
    "Region" nvarchar (15) NULL ,
    "PostalCode" nvarchar (10) NULL ,
    "Country" nvarchar (15) NULL ,
    "Phone" nvarchar (24) NULL ,
    "Fax" nvarchar (24) NULL ,
    "HomePage" "ntext" NULL ,
    CONSTRAINT "PK_Suppliers" PRIMARY KEY  CLUSTERED
    (
        "SupplierID"
    )
)
GO
 CREATE  INDEX "CompanyName" ON "dbo"."Suppliers"("CompanyName")
GO
 CREATE  INDEX "PostalCode" ON "dbo"."Suppliers"("PostalCode")
GO
```

Region

```
CREATE TABLE [dbo].[Region]
    ( [RegionID] [int] NOT NULL ,
    [RegionDescription] [nchar] (50) NOT NULL
) ON [PRIMARY]
GO
```

Orders

```sql
CREATE TABLE "Orders" (
    "OrderID" "int" IDENTITY (1, 1) NOT NULL ,
    "CustomerID" nchar (5) NULL ,
    "EmployeeID" "int" NULL ,
    "OrderDate" "datetime" NULL ,
    "RequiredDate" "datetime" NULL ,
    "ShippedDate" "datetime" NULL ,
    "ShipVia" "int" NULL ,
    "Freight" "money" NULL CONSTRAINT "DF_Orders_Freight" DEFAULT (0),
    "ShipName" nvarchar (40) NULL ,
    "ShipAddress" nvarchar (60) NULL ,
    "ShipCity" nvarchar (15) NULL ,
    "ShipRegion" nvarchar (15) NULL ,
    "ShipPostalCode" nvarchar (10) NULL ,
    "ShipCountry" nvarchar (15) NULL ,
    CONSTRAINT "PK_Orders" PRIMARY KEY  CLUSTERED
    (
        "OrderID"
    ),
    CONSTRAINT "FK_Orders_Customers" FOREIGN KEY
    (
        "CustomerID"
    ) REFERENCES "dbo"."Customers" (
        "CustomerID"
    ),
    CONSTRAINT "FK_Orders_Employees" FOREIGN KEY
    (
        "EmployeeID"
    ) REFERENCES "dbo"."Employees" (
        "EmployeeID"
    ),
    CONSTRAINT "FK Orders Shippers" FOREIGN KEY

    (
        "ShipVia"
    ) REFERENCES "dbo"."Shippers" (
        "ShipperID"
    )
)
GO
 CREATE  INDEX "CustomerID" ON "dbo"."Orders"("CustomerID")
GO
 CREATE  INDEX "CustomersOrders" ON "dbo"."Orders"("CustomerID")
GO
 CREATE  INDEX "EmployeeID" ON "dbo"."Orders"("EmployeeID")
GO
 CREATE  INDEX "EmployeesOrders" ON "dbo"."Orders"("EmployeeID")
GO
 CREATE  INDEX "OrderDate" ON "dbo"."Orders"("OrderDate")
GO
 CREATE  INDEX "ShippedDate" ON "dbo"."Orders"("ShippedDate")
GO
 CREATE  INDEX "ShippersOrders" ON "dbo"."Orders"("ShipVia")
GO
 CREATE  INDEX "ShipPostalCode" ON "dbo"."Orders"("ShipPostalCode")
GO
```

Products

```sql
CREATE TABLE "Products" (
    "ProductID" "int" IDENTITY (1, 1) NOT NULL ,
    "ProductName" nvarchar (40) NOT NULL ,
    "SupplierID" "int" NULL ,
    "CategoryID" "int" NULL ,
    "QuantityPerUnit" nvarchar (20) NULL ,
    "UnitPrice" "money" NULL CONSTRAINT "DF_Products_UnitPrice" DEFAULT (0),
    "UnitsInStock" "smallint" NULL CONSTRAINT "DF_Products_UnitsInStock" DEFAULT (0),
    "UnitsOnOrder" "smallint" NULL CONSTRAINT "DF_Products_UnitsOnOrder" DEFAULT (0),
    "ReorderLevel" "smallint" NULL CONSTRAINT "DF_Products_ReorderLevel" DEFAULT (0),
    "Discontinued" "bit" NOT NULL CONSTRAINT "DF_Products_Discontinued" DEFAULT (0),
    CONSTRAINT "PK_Products" PRIMARY KEY  CLUSTERED
    (
        "ProductID"
    ),
    CONSTRAINT "FK_Products_Categories" FOREIGN KEY
    (
        "CategoryID"
    ) REFERENCES "dbo"."Categories" (
        "CategoryID"
    ),
    CONSTRAINT "FK_Products_Suppliers" FOREIGN KEY
    (
        "SupplierID"
    ) REFERENCES "dbo"."Suppliers" (
        "SupplierID"
    ),
    CONSTRAINT "CK_Products_UnitPrice" CHECK (UnitPrice >= 0),
    CONSTRAINT "CK_ReorderLevel" CHECK (ReorderLevel >= 0),
    CONSTRAINT "CK_UnitsInStock" CHECK (UnitsInStock >= 0),
    CONSTRAINT "CK_UnitsOnOrder" CHECK (UnitsOnOrder >= 0)
)
GO

 CREATE  INDEX "CategoriesProducts" ON "dbo"."Products"("CategoryID")
GO
 CREATE  INDEX "CategoryID" ON "dbo"."Products"("CategoryID")
GO
 CREATE  INDEX "ProductName" ON "dbo"."Products"("ProductName")
GO
 CREATE  INDEX "SupplierID" ON "dbo"."Products"("SupplierID")
GO
 CREATE  INDEX "SuppliersProducts" ON "dbo"."Products"("SupplierID")
GO
```

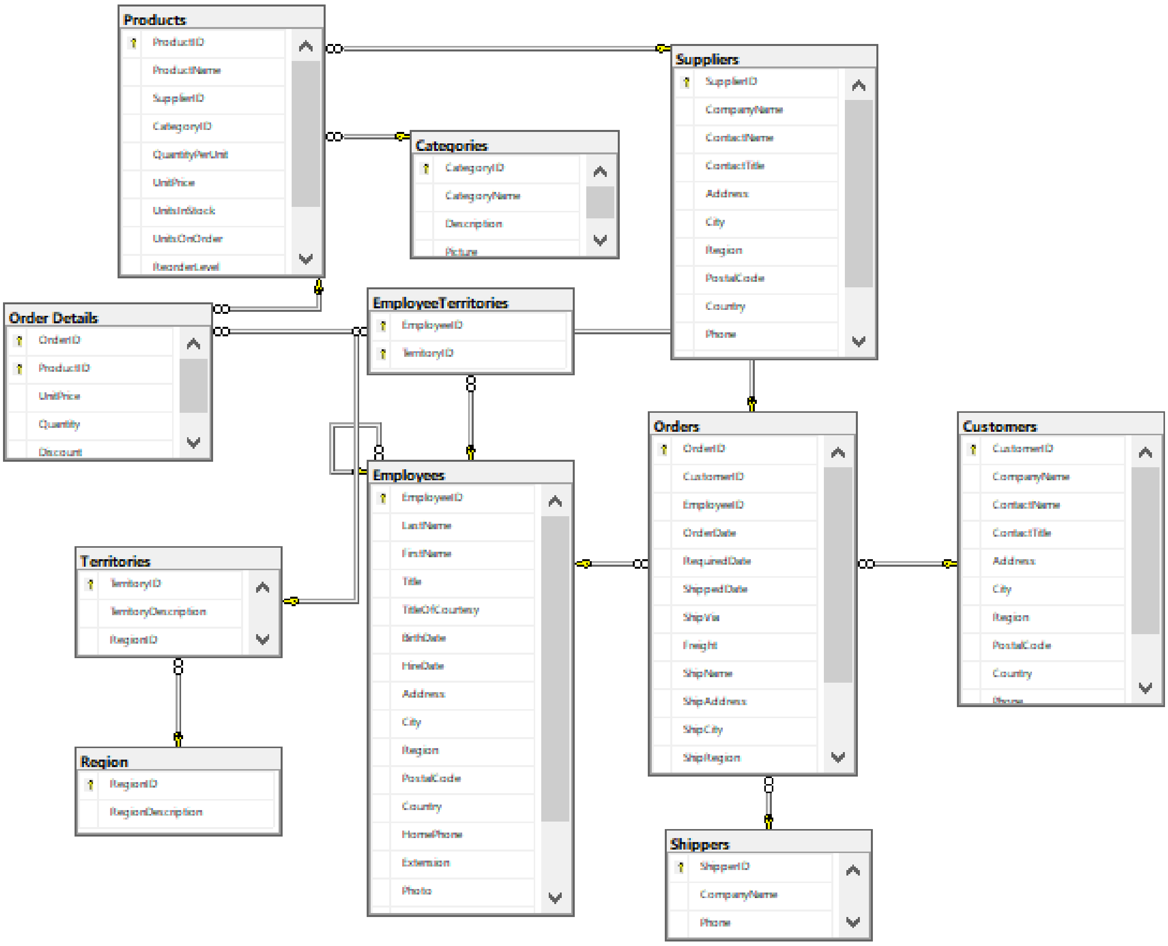
Order Details

```sql
CREATE TABLE "Order Details" (
    "OrderID" "int" NOT NULL ,
    "ProductID" "int" NOT NULL ,
    "UnitPrice" "money" NOT NULL CONSTRAINT "DF_Order_Details_UnitPrice" DEFAULT (0),
    "Quantity" "smallint" NOT NULL CONSTRAINT "DF_Order_Details_Quantity" DEFAULT (1),
    "Discount" "real" NOT NULL CONSTRAINT "DF_Order_Details_Discount" DEFAULT (0),
    CONSTRAINT "PK_Order_Details" PRIMARY KEY  CLUSTERED
    (
        "OrderID",
        "ProductID"
    ),
    CONSTRAINT "FK_Order_Details_Orders" FOREIGN KEY
    (
        "OrderID"
    ) REFERENCES "dbo"."Orders" (
        "OrderID"
    ),
    CONSTRAINT "FK_Order_Details_Products" FOREIGN KEY
    (
        "ProductID"
    ) REFERENCES "dbo"."Products" (
        "ProductID"
    ),
    CONSTRAINT "CK_Discount" CHECK (Discount >= 0 and (Discount <= 1)),
    CONSTRAINT "CK_Quantity" CHECK (Quantity > 0),
    CONSTRAINT "CK_UnitPrice" CHECK (UnitPrice >= 0)
)
GO
 CREATE  INDEX "OrderID" ON "dbo"."Order Details"("OrderID")
GO
 CREATE  INDEX "OrdersOrder_Details" ON "dbo"."Order Details"("OrderID")
GO
 CREATE  INDEX "ProductID" ON "dbo"."Order Details"("ProductID")

 CREATE  INDEX "ProductsOrder_Details" ON "dbo"."Order Details"("ProductID")
GO
```

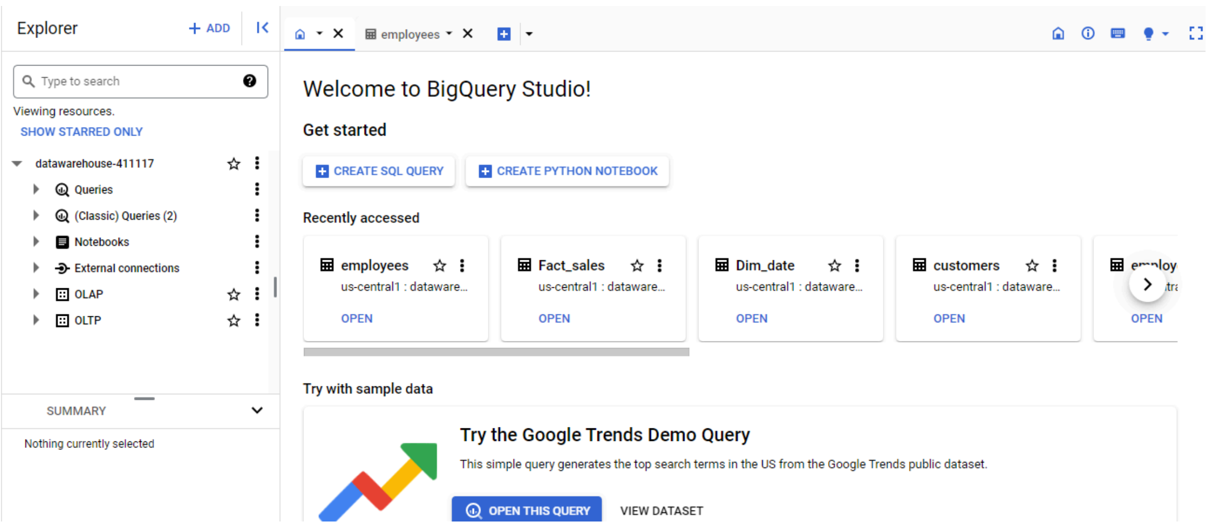
Territories and EmployeeTerritories:

```sql
CREATE TABLE [dbo].[Territories]
    ([TerritoryID] [nvarchar] (20) NOT NULL ,
    [TerritoryDescription] [nchar] (50) NOT NULL ,
        [RegionID] [int] NOT NULL
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[EmployeeTerritories]
    ([EmployeeID] [int] NOT NULL,
    [TerritoryID] [nvarchar] (20) NOT NULL
) ON [PRIMARY]
```

3. Schema of tables



4. Configure a specific Data Warehouse system (Google Bigquery)



III. Build a data warehouse

1. Choose the Business Process

Business Process: Sales Analysis

In this case, the primary focus is on understanding and improving the sales performance of products. This involves analyzing which products are sold the most, where they are sold, and which products generate the highest revenue.

2. **Declare the Grain**

The grain is known as the lowest level of the star schema that a business would have. The grain sets the foundation of the entire model development from the order and the inventory business drivers. It determines the level of detailed information that can be made available to the dimensional model. It is quite essential to give this section more time and accurate results because any error on the grain section can cause redesigning of the whole model. The table's grain consists of information on the facts in the data, necessary when building a report for business intelligence. The level at which a row is granular should be atomic, and it holds the most specific details for tables in any data warehouse. Atomic grain refers to the lowest level of grain that is possibly captured by a business on daily and monthly aggregations of individual transactions.

In this case, the grain should be at the transactional level, capturing individual sales transactions, is sales information to understand the number of orders, number of products, number of customers, number of sales staff and the date.

Components of the Grain:

- orderID (Primary Key): A unique identifier for each transaction. This ensures that each record in the data warehouse is distinct and can be easily referenced.
- productID (Foreign Key): A reference to the Product Dimension, linking the sale to the specific product that was sold. This enables analysis at the product level.
- customerID (Foreign Key): A reference to the Customer Dimension, linking the sale to the customer who made the purchase. This facilitates customer-centric analysis, such as identifying the top customers or analyzing sales by customer segments.
- employeeID (Foreign Key): A reference to the Employee Dimension, linking the sale to the employee, analysis of sales performance on an individual employee level.
- categoryID (Foreign Key): A reference to the Category Dimension, linking the sale to the category of product.
- date_key (Foreign Key): A reference to the Date Dimension, indicating the date and time of the sale. This allows for time-based analysis, such as sales trends over days, months, and years.OrderID

3. **Identify the Dimensions**

The Customers dimension table is a copy of the Customers table and contains an extra attribute: Insertion_timestamp.

The Employees dimension table is a copy of the Employees table and contains an extra attribute: Insertion_timestamp.

The Products dimension table is a copy of the Products table and contains an extra attribute: Insertion_timestamp.

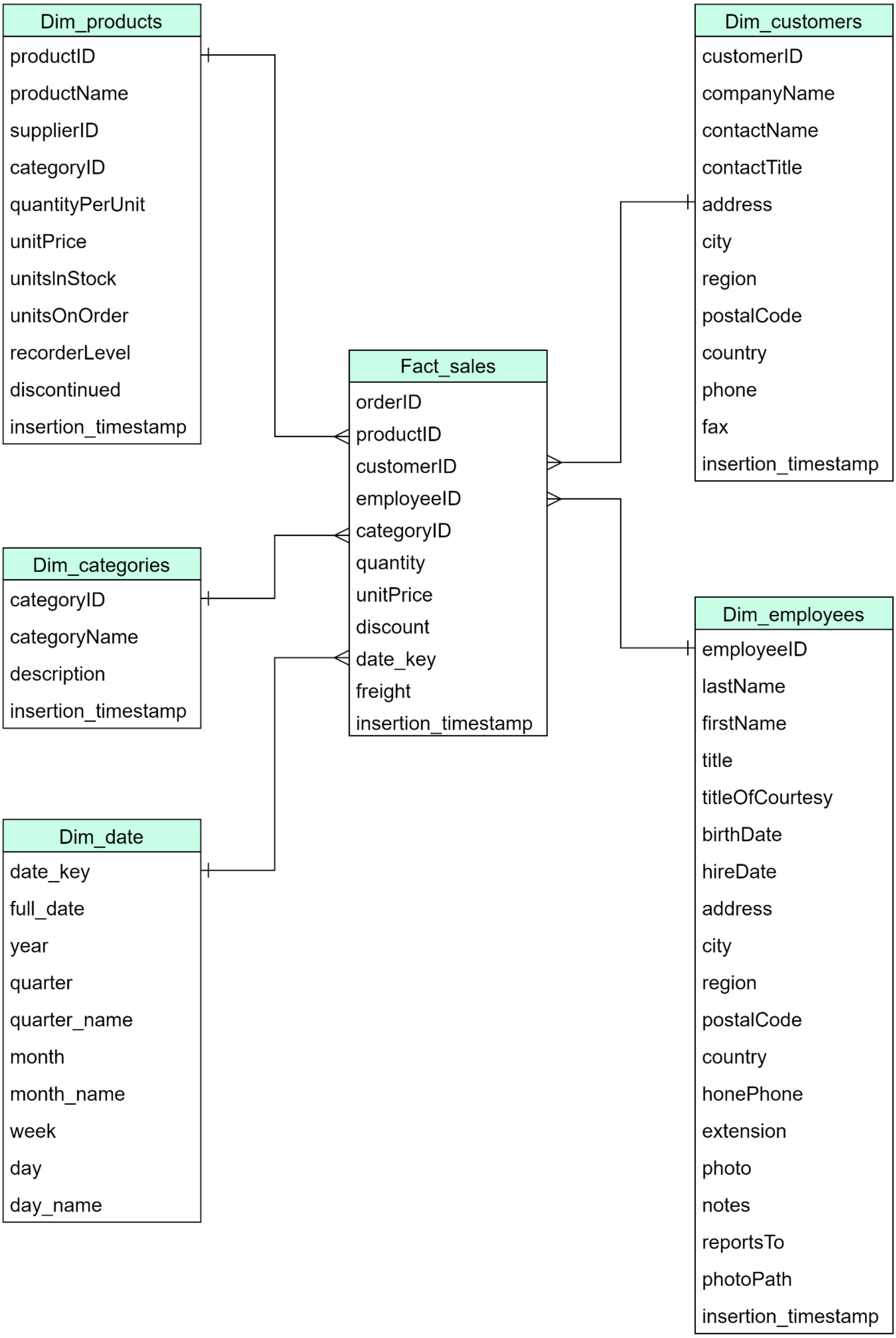
The Categories dimension table is created from the Categories table by cutting out attributes: Picture and contains an extra attribute: Insertion_timestamp.

The Date dimension table includes attributes: Date_key, Full_date, Year, Quarter, Quarter_name, Month, Month_name, Week, Day, Day_name.

4. **Identify the Fact**

After identifying the dimensions, one must determine the fact tables. Facts are measurable data such as prices per unit, which are in a fact table. The facts are orders and order details. Variables are unit price, quantity, discount, units on stock.

5. Build the schema



IV. Deployment

1. Yml file Docker:

```
# Licensed to the Apache Software Foundation (ASF) under one
# or more contributor license agreements.  See the NOTICE file
# distributed with this work for additional information
# regarding copyright ownership.  The ASF licenses this file
# to you under the Apache License, Version 2.0 (the
# "License"); you may not use this file except in compliance
# with the License.  You may obtain a copy of the License at
#
#   http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing,
# software distributed under the License is distributed on an
# "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
# KIND, either express or implied.  See the License for the
# specific language governing permissions and limitations
# under the License.
#

# Basic Airflow cluster configuration for CeleryExecutor with Redis and
PostgreSQL.
#
# WARNING: This configuration is for local development. Do not use it
in a production deployment.
#
# This configuration supports basic configuration using environment
variables or an .env file
# The following variables are supported:
#
# AIRFLOW_IMAGE_NAME           - Docker image name used to run Airflow.
#                                Default: apache/airflow:2.8.0
# AIRFLOW_UID                  - User ID in Airflow containers
#                                Default: 50000
# AIRFLOW_PROJ_DIR             - Base path to which all the files will
be volumed.
#                                Default: .
# Those configurations are useful mostly in case of standalone
testing/running Airflow in test/try-out mode
#
# _AIRFLOW_WWW_USER_USERNAME   - Username for the administrator account
(if requested).
#                                Default: airflow
```

```
# _AIRFLOW_WWW_USER_PASSWORD   - Password for the administrator account
(if requested).
#                                Default: airflow
# _PIP_ADDITIONAL_REQUIREMENTS  - Additional PIP requirements to add
when starting all containers.
#                                Use this option ONLY for quick checks.
Installing requirements at container
#                                startup is done EVERY TIME the service
is started.
#                                    A better way is to build a custom
image or extend the official image
#                                                     as described in
https://airflow.apache.org/docs/docker-stack/build.html.
#                                Default: ''
#
# Feel free to modify this file to suit your needs.
---
x-airflow-common:
  &airflow-common
   # In order to add custom dependencies or upgrade provider packages
you can use your extended image.
   # Comment the image line, place your Dockerfile in the directory
where you placed the docker-compose.yaml
   # and uncomment the "build" line below, Then run `docker-compose
build` to build the images.
  image: ${AIRFLOW_IMAGE_NAME:-apache/airflow:2.8.0}
  # build: .
  environment:
    &airflow-common-env
    AIRFLOW__CORE__EXECUTOR: CeleryExecutor
                                   AIRFLOW__DATABASE__SQL_ALCHEMY_CONN:
postgresql+psycopg2://airflow:airflow@postgres/airflow
                                       AIRFLOW__CELERY__RESULT_BACKEND:
db+postgresql://airflow:airflow@postgres/airflow
    AIRFLOW__CELERY__BROKER_URL: redis://:@redis:6379/0
    AIRFLOW__CORE__FERNET_KEY: ''
    AIRFLOW__CORE__DAGS_ARE_PAUSED_AT_CREATION: 'true'
    AIRFLOW__CORE__LOAD_EXAMPLES: 'true'
                                           AIRFLOW__API__AUTH_BACKENDS:
'airflow.api.auth.backend.basic_auth,airflow.api.auth.backend.session'
    # yamllint disable rule:line-length
    # Use simple http server on scheduler for health checks
```

```yaml
                                                      # See
https://airflow.apache.org/docs/apache-airflow/stable/administration-an
d-deployment/logging-monitoring/check-health.html#scheduler-health-chec
k-server
    # yamllint enable rule:line-length
    AIRFLOW__SCHEDULER__ENABLE_HEALTH_CHECK: 'true'
    # WARNING: Use _PIP_ADDITIONAL_REQUIREMENTS option ONLY for a quick
checks
    # for other purpose (development, test and especially production
usage) build/extend Airflow image.
    _PIP_ADDITIONAL_REQUIREMENTS: ${_PIP_ADDITIONAL_REQUIREMENTS:-}
  volumes:
    - ${AIRFLOW_PROJ_DIR:-.}/dags:/opt/airflow/dags
    - ${AIRFLOW_PROJ_DIR:-.}/logs:/opt/airflow/logs
    - ${AIRFLOW_PROJ_DIR:-.}/config:/opt/airflow/config
    - ${AIRFLOW_PROJ_DIR:-.}/plugins:/opt/airflow/plugins
                                                                       -
~/.config/gcloud/application_default_credentials.json:/home/airflow/.co
nfig/gcloud/application_default_credentials.json
  user: "${AIRFLOW_UID:-50000}:0"
  depends_on:
    &airflow-common-depends-on
    redis:
      condition: service_healthy
    postgres:
      condition: service_healthy

services:
  postgres:
    image: postgres:13
    environment:
      POSTGRES_USER: airflow
      POSTGRES_PASSWORD: airflow
      POSTGRES_DB: airflow
    volumes:
      - postgres-db-volume:/var/lib/postgresql/data
    healthcheck:
      test: ["CMD", "pg_isready", "-U", "airflow"]
      interval: 10s
      retries: 5
      start_period: 5s
    restart: always
```

```yaml
  redis:
    image: redis:latest
    expose:
      - 6379
    healthcheck:
      test: ["CMD", "redis-cli", "ping"]
      interval: 10s
      timeout: 30s
      retries: 50
      start_period: 30s
    restart: always

  airflow-webserver:
    <<: *airflow-common
    command: webserver
    ports:
      - "8080:8080"
    healthcheck:
      test: ["CMD", "curl", "--fail", "http://localhost:8080/health"]
      interval: 30s
      timeout: 10s
      retries: 5
      start_period: 30s
    restart: always
    depends_on:
      <<: *airflow-common-depends-on
      airflow-init:
        condition: service_completed_successfully

  airflow-scheduler:
    <<: *airflow-common
    command: scheduler
    healthcheck:
      test: ["CMD", "curl", "--fail", "http://localhost:8974/health"]
      interval: 30s
      timeout: 10s
      retries: 5
      start_period: 30s
    restart: always
    depends_on:
      <<: *airflow-common-depends-on
      airflow-init:
        condition: service_completed_successfully
```

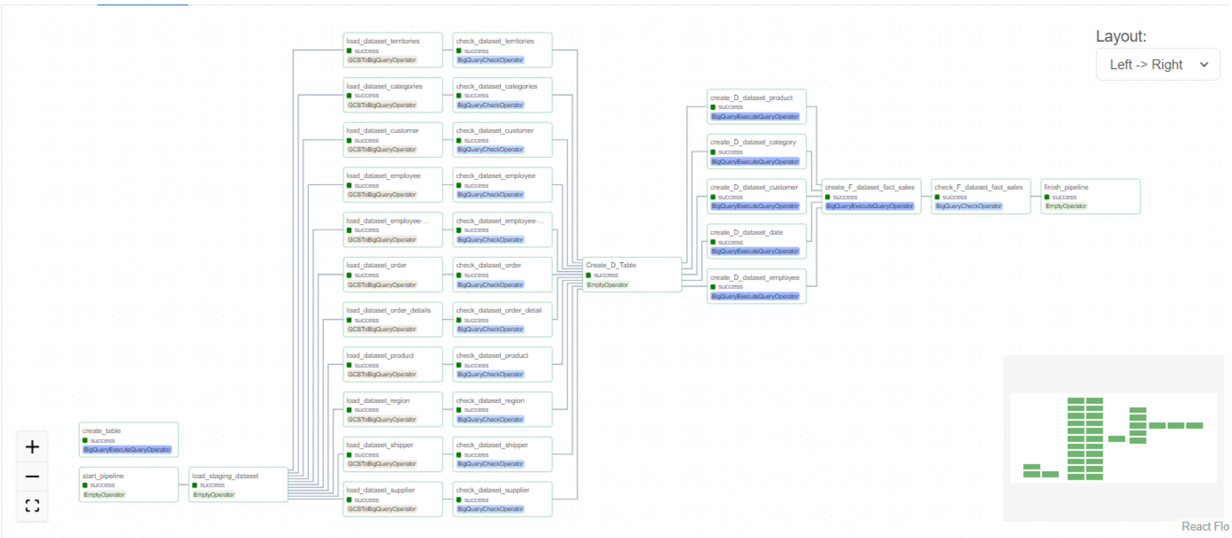
```yaml
  airflow-worker:
    <<: *airflow-common
    command: celery worker
    healthcheck:
      # yamllint disable rule:line-length
      test:
        - "CMD-SHELL"
        - 'celery --app airflow.providers.celery.executors.celery_executor.app inspect ping -d "celery@$${HOSTNAME}" || celery --app airflow.executors.celery_executor.app inspect ping -d "celery@$${HOSTNAME}"'
      interval: 30s
      timeout: 10s
      retries: 5
      start_period: 30s
    environment:
      <<: *airflow-common-env
      # Required to handle warm shutdown of the celery workers properly
      # See https://airflow.apache.org/docs/docker-stack/entrypoint.html#signal-propagation
      DUMB_INIT_SETSID: "0"
    restart: always
    depends_on:
      <<: *airflow-common-depends-on
      airflow-init:
        condition: service_completed_successfully

  airflow-triggerer:
    <<: *airflow-common
    command: triggerer
    healthcheck:
      test: ["CMD-SHELL", 'airflow jobs check --job-type TriggererJob --hostname "$${HOSTNAME}"']
      interval: 30s
      timeout: 10s
      retries: 5
      start_period: 30s
    restart: always
    depends_on:
      <<: *airflow-common-depends-on
```

```
      airflow-init:
        condition: service_completed_successfully

  airflow-init:
    <<: *airflow-common
    entrypoint: /bin/bash
    # yamllint disable rule:line-length
    command:
      - -c
      - |
        if [[ -z "${AIRFLOW_UID}" ]]; then
          echo
          echo -e "\033[1;33mWARNING!!!: AIRFLOW_UID not set!\e[0m"
          echo "If you are on Linux, you SHOULD follow the instructions
below to set "
          echo "AIRFLOW_UID environment variable, otherwise files will
be owned by root."
          echo "For other operating systems you can get rid of the
warning with manually created .env file:"
          echo "    See:
https://airflow.apache.org/docs/apache-airflow/stable/howto/docker-comp
ose/index.html#setting-the-right-airflow-user"
          echo
        fi
        one_meg=1048576
        mem_available=$$(($$(getconf _PHYS_PAGES) * $$(getconf
PAGE_SIZE) / one_meg))
        cpus_available=$$(grep -cE 'cpu[0-9]+' /proc/stat)
        disk_available=$$(df / | tail -1 | awk '{print $$4}')
        warning_resources="false"
        if (( mem_available < 4000 )) ; then
          echo
          echo -e "\033[1;33mWARNING!!!: Not enough memory available
for Docker.\e[0m"
          echo "At least 4GB of memory required. You have $$(numfmt
--to iec $$((mem_available * one_meg)))"
          echo
          warning_resources="true"
        fi
        if (( cpus_available < 2 )); then
          echo
          echo -e "\033[1;33mWARNING!!!: Not enough CPUS available for
Docker.\e[0m"
```

```
                      echo  "At  least  2  CPUs  recommended.  You  have
$${cpus_available}"
          echo
          warning_resources="true"
        fi
        if (( disk_available < one_meg * 10 )); then
          echo
                echo  -e  "\033[1;33mWARNING!!!:  Not  enough  Disk  space
available for Docker.\e[0m"
            echo "At least 10 GBs recommended. You have $$(numfmt --to
iec $$((disk_available * 1024 )))"
          echo
          warning_resources="true"
        fi
        if [[ $${warning_resources} == "true" ]]; then
          echo
           echo -e "\033[1;33mWARNING!!!: You have not enough resources
to run Airflow (see above)!\e[0m"
            echo "Please follow the instructions to increase amount of
resources available:"
                                                              echo      "
https://airflow.apache.org/docs/apache-airflow/stable/howto/docker-comp
ose/index.html#before-you-begin"
          echo
        fi
        mkdir -p /sources/logs /sources/dags /sources/plugins
        chown -R "${AIRFLOW_UID}:0" /sources/{logs,dags,plugins}
        exec /entrypoint airflow version
    # yamllint enable rule:line-length
    environment:
      <<: *airflow-common-env
      _AIRFLOW_DB_MIGRATE: 'true'
      _AIRFLOW_WWW_USER_CREATE: 'true'
                                             _AIRFLOW_WWW_USER_USERNAME:
${_AIRFLOW_WWW_USER_USERNAME:-airflow}
                                             _AIRFLOW_WWW_USER_PASSWORD:
${_AIRFLOW_WWW_USER_PASSWORD:-airflow}
      _PIP_ADDITIONAL_REQUIREMENTS: ''
    user: "0:0"
    volumes:
      - ${AIRFLOW_PROJ_DIR:-.}:/sources

  airflow-cli:
```

```
    <<: *airflow-common
    profiles:
      - debug
    environment:
      <<: *airflow-common-env
      CONNECTION_CHECK_MAX_COUNT: "0"
                 #    Workaround    for    entrypoint    issue.    See:
https://github.com/apache/airflow/issues/16252
    command:
      - bash
      - -c
      - airflow

   #  You  can  enable  flower  by  adding  "--profile  flower"  option e.g.
docker-compose --profile flower up
   # or by explicitly targeted on the command line e.g. docker-compose
up flower.
  # See: https://docs.docker.com/compose/profiles/
  flower:
    <<: *airflow-common
    command: celery flower
    profiles:
      - flower
    ports:
      - "5555:5555"
    healthcheck:
      test: ["CMD", "curl", "--fail", "http://localhost:5555/"]
      interval: 30s
      timeout: 10s
      retries: 5
      start_period: 30s
    restart: always
    depends_on:
      <<: *airflow-common-depends-on
      airflow-init:
        condition: service_completed_successfully

volumes:
  postgres-db-volume:
```

2. **DAG file:**



Building a pipeline transform OLTP to OLAP

```
from datetime import timedelta, datetime
from airflow import DAG
from airflow.utils.dates import days_ago
from airflow.operators.dummy_operator import DummyOperator
from airflow.providers.google.cloud.transfers.gcs_to_bigquery import GCSToBigQueryOperator
from airflow.providers.google.cloud.operators.bigquery import BigQueryCheckOperator
from airflow.contrib.operators.bigquery_operator import BigQueryOperator


GOOGLE_CONN_ID = "google_cloud_default"
PROJECT_ID="datawarehouse-411117"
GS_PATH = "data/"
BUCKET_NAME = 'us-central1-datawarehouse-ad43d7dc-bucket'
STAGING_DATASET = "OLTP"
DATASET = "OLAP"
LOCATION = "us-central1"

default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': datetime(2024, 1, 1),
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
```

```
}

with        DAG('northwind',        schedule_interval=timedelta(days=1),
default_args=default_args) as dag:
    start_pipeline = DummyOperator(
        task_id = 'start_pipeline',
        dag = dag
        )


    load_staging_dataset = DummyOperator(
        task_id = 'load_staging_dataset',
        dag = dag
        )
    create_table_task = BigQueryOperator(
        task_id='create_table',
        sql="""
        CREATE OR REPLACE TABLE `datawarehouse-411117.OLTP.customers` (
            customerID STRING,
            companyName STRING,
            contactName STRING,
            contactTitle STRING,
            address STRING,
            city STRING,
            region STRING,
            postalCode STRING,
            country STRING,
            phone STRING,
            fax STRING
        )
        """,
    use_legacy_sql=False,
    location='us-central1',
    dag=dag,
)

    load_dataset_customer = GCSToBigQueryOperator(
        task_id='load_dataset_customer',
        bucket=BUCKET_NAME,
        source_objects=['data/customers.csv'],

destination_project_dataset_table=f'{PROJECT_ID}:{STAGING_DATASET}.cust
omers',
```

```
        write_disposition='WRITE_TRUNCATE',
        source_format='CSV',
        field_delimiter=',',
        schema_fields=[
                      {'name':  'customerID',  'type':  'STRING',  'mode':
'NULLABLE'},
                      {'name':  'companyName',  'type':  'STRING',  'mode':
'NULLABLE'},
                      {'name':  'contactName',  'type':  'STRING',  'mode':
'NULLABLE'},
                      {'name':  'contactTitle',  'type':  'STRING',  'mode':
'NULLABLE'},
            {'name': 'address', 'type': 'STRING', 'mode': 'NULLABLE'},
            {'name': 'city', 'type': 'STRING', 'mode': 'NULLABLE'},
            {'name': 'region', 'type': 'STRING', 'mode': 'NULLABLE'},
                      {'name':  'postalCode',  'type':  'STRING',  'mode':
'NULLABLE'},
            {'name': 'country', 'type': 'STRING', 'mode': 'NULLABLE'},
            {'name': 'phone', 'type': 'STRING', 'mode': 'NULLABLE'},
            {'name': 'fax', 'type': 'STRING', 'mode': 'NULLABLE'},
        ],
    )



    load_dataset_employee = GCSToBigQueryOperator(
        task_id = 'load_dataset_employee',
        bucket = BUCKET_NAME,
        source_objects = ['data/employees.csv'],
                                  destination_project_dataset_table    =
f'{PROJECT_ID}:{STAGING_DATASET}.employees',
        write_disposition='WRITE_TRUNCATE',
        source_format = 'csv',
        field_delimiter=',',
    )

    load_dataset_product = GCSToBigQueryOperator(
        task_id = 'load_dataset_product',
        bucket = BUCKET_NAME,
        source_objects = ['data/products.csv'],
                                  destination_project_dataset_table    =
f'{PROJECT_ID}:{STAGING_DATASET}.products',
        write_disposition='WRITE_TRUNCATE',
```

```
        source_format = 'csv',
        field_delimiter=',',
    )

    load_dataset_supplier = GCSToBigQueryOperator(
        task_id = 'load_dataset_supplier',
        bucket = BUCKET_NAME,
        source_objects = ['data/suppliers.csv'],
                                   destination_project_dataset_table     =
f'{PROJECT_ID}:{STAGING_DATASET}.suppliers',
        write_disposition='WRITE_TRUNCATE',
        source_format = 'csv',
        field_delimiter=',',
    )
    load_dataset_order = GCSToBigQueryOperator(
        task_id = 'load_dataset_order',
        bucket = BUCKET_NAME,
        source_objects = ['data/orders.csv'],
                                   destination_project_dataset_table     =
f'{PROJECT_ID}:{STAGING_DATASET}.orders',
        write_disposition='WRITE_TRUNCATE',
        source_format = 'csv',
        field_delimiter=',',
    )
    load_dataset_order_details = GCSToBigQueryOperator(
        task_id = 'load_dataset_order_details',
        bucket = BUCKET_NAME,
        source_objects = ['data/order-details.csv'],
                                   destination_project_dataset_table     =
f'{PROJECT_ID}:{STAGING_DATASET}.order-details',
        write_disposition='WRITE_TRUNCATE',
        source_format = 'csv',
        field_delimiter=',',
    )
    load_dataset_categories = GCSToBigQueryOperator(
        task_id = 'load_dataset_categories',
        bucket = BUCKET_NAME,
        source_objects = ['data/categories.csv'],
                                   destination_project_dataset_table     =
f'{PROJECT_ID}:{STAGING_DATASET}.categories',
        write_disposition='WRITE_TRUNCATE',
        source_format = 'csv',
        field_delimiter=',',
```

```
    )
    load_dataset_territories = GCSToBigQueryOperator(
        task_id = 'load_dataset_territories',
        bucket = BUCKET_NAME,
        source_objects = ['data/territories.csv'],
                                 destination_project_dataset_table    =
f'{PROJECT_ID}:{STAGING_DATASET}.territories',
        write_disposition='WRITE_TRUNCATE',
        source_format = 'csv',
        field_delimiter=',',
    )
    load_dataset_employee_territories = GCSToBigQueryOperator(
        task_id = 'load_dataset_employee-territories',
        bucket = BUCKET_NAME,
        source_objects = ['data/employee-territories.csv'],
                                 destination_project_dataset_table    =
f'{PROJECT_ID}:{STAGING_DATASET}.employee-territories',
        write_disposition='WRITE_TRUNCATE',
        source_format = 'csv',
        field_delimiter=',',
    )
    load_dataset_region = GCSToBigQueryOperator(
        task_id = 'load_dataset_region',
        bucket = BUCKET_NAME,
        source_objects = ['data/regions.csv'],
                                 destination_project_dataset_table    =
f'{PROJECT_ID}:{STAGING_DATASET}.regions',
        write_disposition='WRITE_TRUNCATE',
        source_format = 'csv',
        field_delimiter=',',
    )
    load_dataset_shipper = GCSToBigQueryOperator(
        task_id = 'load_dataset_shipper',
        bucket = BUCKET_NAME,
        source_objects = ['data/shippers.csv'],
                                 destination_project_dataset_table    =
f'{PROJECT_ID}:{STAGING_DATASET}.shippers',
        write_disposition='WRITE_TRUNCATE',
        source_format = 'csv',
        field_delimiter=',',
    )
    check_dataset_customer = BigQueryCheckOperator(
        task_id = 'check_dataset_customer',
```

```
        use_legacy_sql=False,
        location = LOCATION,
                                sql     =     f'SELECT     count(*)     FROM
`{PROJECT_ID}.{STAGING_DATASET}.customers`'
        )

    check_dataset_employee = BigQueryCheckOperator(
        task_id = 'check_dataset_employee',
        use_legacy_sql=False,
        location = LOCATION,
                                sql     =     f'SELECT     count(*)     FROM
`{PROJECT_ID}.{STAGING_DATASET}.employees`'
        )

    check_dataset_product = BigQueryCheckOperator(
        task_id = 'check_dataset_product',
        use_legacy_sql=False,
        location = LOCATION,
                                sql     =     f'SELECT     count(*)     FROM
`{PROJECT_ID}.{STAGING_DATASET}.products`'
        )
    check_dataset_supplier = BigQueryCheckOperator(
        task_id = 'check_dataset_supplier',
        use_legacy_sql=False,
        location = LOCATION,
                                sql     =     f'SELECT     count(*)     FROM
`{PROJECT_ID}.{STAGING_DATASET}.suppliers`'
        )
    check_dataset_order = BigQueryCheckOperator(
        task_id = 'check_dataset_order',
        use_legacy_sql=False,
        location = LOCATION,
                                sql     =     f'SELECT     count(*)     FROM
`{PROJECT_ID}.{STAGING_DATASET}.orders`'
        )
    check_dataset_order_detail = BigQueryCheckOperator(
        task_id = 'check_dataset_order_detail',
        use_legacy_sql=False,
        location = LOCATION,
                                sql     =     f'SELECT     count(*)     FROM
`{PROJECT_ID}.{STAGING_DATASET}.order-details`'
        )
    check_dataset_categories = BigQueryCheckOperator(
```

```
        task_id = 'check_dataset_categories',
        use_legacy_sql=False,
        location = LOCATION,
                                sql    =    f'SELECT    count(*)    FROM
`{PROJECT_ID}.{STAGING_DATASET}.categories`'
        )
    check_dataset_employee_territories = BigQueryCheckOperator(
        task_id = 'check_dataset_employee-territories',
        use_legacy_sql=False,
        location = LOCATION,
                                sql    =    f'SELECT    count(*)    FROM
`{PROJECT_ID}.{STAGING_DATASET}.employee-territories`'
        )
    check_dataset_territories = BigQueryCheckOperator(
        task_id = 'check_dataset_territories',
        use_legacy_sql=False,
        location = LOCATION,
                                sql    =    f'SELECT    count(*)    FROM
`{PROJECT_ID}.{STAGING_DATASET}.territories`'
        )
    check_dataset_region = BigQueryCheckOperator(
        task_id = 'check_dataset_region',
        use_legacy_sql=False,
        location = LOCATION,
                                sql    =    f'SELECT    count(*)    FROM
`{PROJECT_ID}.{STAGING_DATASET}.regions`'
        )
    check_dataset_shipper = BigQueryCheckOperator(
        task_id = 'check_dataset_shipper',
        use_legacy_sql=False,
        location = LOCATION,
                                sql    =    f'SELECT    count(*)    FROM
`{PROJECT_ID}.{STAGING_DATASET}.shippers`'
        )
    create_D_Table = DummyOperator(
        task_id = 'Create_D_Table',
        dag = dag
        )

    create_D_dataset_customer = BigQueryOperator(
        task_id = 'create_D_dataset_customer',
        use_legacy_sql = False,
        location = LOCATION,
```

```
        sql = './sql/dim_customers.sql'
        )

    create_D_dataset_employee = BigQueryOperator(
        task_id = 'create_D_dataset_employee',
        use_legacy_sql = False,
        location = LOCATION,
        sql = './sql/dim_employees.sql'
        )

    create_D_dataset_product = BigQueryOperator(
        task_id = 'create_D_dataset_product',
        use_legacy_sql = False,
        location = LOCATION,
        sql = './sql/dim_products.sql'
        )
    create_D_dataset_category = BigQueryOperator(
        task_id = 'create_D_dataset_category',
        use_legacy_sql = False,
        location = LOCATION,
        sql = './sql/dim_category.sql'
        )
    create_D_dataset_date = BigQueryOperator(
        task_id = 'create_D_dataset_date',
        use_legacy_sql = False,
        location = LOCATION,
        sql = './sql/dim_date.sql'
        )

    create_F_dataset_fact_sales = BigQueryOperator(
        task_id = 'create_F_dataset_fact_sales',
        use_legacy_sql = False,
        location = LOCATION,
        sql = './sql/fact_sales.sql'
        )
    check_F_dataset_fact_sales = BigQueryCheckOperator(
        task_id = 'check_F_dataset_fact_sales',
        use_legacy_sql=False,
        location = LOCATION,
                                sql     =     f'SELECT     count(*)     FROM
`{PROJECT_ID}.{DATASET}.Fact_sales`'
        )
```

```
    finish_pipeline = DummyOperator(
        task_id = 'finish_pipeline',
        dag = dag
        )
start_pipeline >> load_staging_dataset

load_staging_dataset  >> [load_dataset_customer, load_dataset_employee,
load_dataset_product,load_dataset_supplier,load_dataset_order,
load_dataset_order_details,
load_dataset_categories,load_dataset_employee_territories,
load_dataset_territories, load_dataset_region, load_dataset_shipper]

load_dataset_customer >> check_dataset_customer
load_dataset_employee >> check_dataset_employee
load_dataset_product >> check_dataset_product
load_dataset_supplier >> check_dataset_supplier
load_dataset_order >> check_dataset_order
load_dataset_order_details >> check_dataset_order_detail
load_dataset_categories >> check_dataset_categories
load_dataset_employee_territories >> check_dataset_employee_territories
load_dataset_territories >>check_dataset_territories
load_dataset_region >>check_dataset_region
load_dataset_shipper >> check_dataset_shipper

[check_dataset_customer, check_dataset_employee, check_dataset_product,
check_dataset_supplier,check_dataset_order,
check_dataset_order_detail,check_dataset_categories,
check_dataset_employee_territories,
check_dataset_territories,check_dataset_region,check_dataset_shipper   ]
>> create_D_Table

create_D_Table>> [create_D_dataset_customer, create_D_dataset_employee,
create_D_dataset_product,create_D_dataset_category,
create_D_dataset_date]

[create_D_dataset_customer,create_D_dataset_employee,
create_D_dataset_product,create_D_dataset_category,
create_D_dataset_date] >> create_F_dataset_fact_sales

create_F_dataset_fact_sales>>check_F_dataset_fact_sales>>
finish_pipeline
```

3. **Transform code content to transform Data Warehouse tables:**

- dags\sql\dim_category.sql:

```
CREATE OR REPLACE TABLE
  `datawarehouse-411117.OLAP.Dim_categories` AS
SELECT
  categoryID,
  categoryName,
  description,
  current_timestamp() as  insertion_timestamp,
FROM
  `datawarehouse-411117.OLTP.categories`
QUALIFY ROW_NUMBER() OVER(PARTITION BY categoryID) = 1;
```

- dags\sql\dim_customers.sql:

```
CREATE OR REPLACE TABLE
  datawarehouse-411117.OLAP.Dim_customers AS
SELECT
  customerID,
  companyName,
  contactName,
  contactTitle,
  address,
  city,
  region,
  postalCode,
  country,
  phone,
  fax,
  current_timestamp() as  insertion_timestamp,
FROM
  datawarehouse-411117.OLTP.customers
QUALIFY ROW_NUMBER() OVER(PARTITION BY customerID) = 1;
```

- dags\sql\dim_date.sql:

```
CREATE OR REPLACE TABLE `datawarehouse-411117.OLAP.Dim_date` AS
SELECT
    CAST(FORMAT_DATE('%Y%m%d', date) AS INT64) AS date_key,
    date AS full_date,
    EXTRACT(YEAR FROM date) AS year,
    EXTRACT(QUARTER FROM date) AS quarter,
```

```sql
       CONCAT('Q',  CAST(EXTRACT(QUARTER  FROM  date)  AS  STRING))  AS
quarter_name,
    EXTRACT(MONTH FROM date) AS month,
    FORMAT_DATE('%B', date) AS month_name,
    EXTRACT(WEEK FROM date) AS week,
    EXTRACT(DAY FROM date) AS day,
    FORMAT_DATE('%A', date) AS day_name
FROM
      UNNEST(GENERATE_DATE_ARRAY(DATE  '1996-07-04',  DATE  '2030-12-31',
INTERVAL 1 DAY)) AS date;
```

- dags\sql\dim_employees.sql:

```sql
CREATE OR REPLACE TABLE
  datawarehouse-411117.OLAP.Dim_employees AS
SELECT
  employeeID,
  lastName,
  firstName,
  title,
  titleOfCourtesy,
  birthDate,
  hireDate,
  address,
  city,
  region,
  postalCode,
  country,
  homePhone,
  extension,
  photo,
  notes,
  reportsTo,
  photoPath,
  current_timestamp() as  insertion_timestamp,
FROM
  datawarehouse-411117.OLTP.employees
QUALIFY ROW_NUMBER() OVER(PARTITION BY employeeID) = 1;
```

- dags\sql\dim_products.sql:

```sql
CREATE OR REPLACE TABLE
  `datawarehouse-411117.OLAP.Dim_products` AS
```

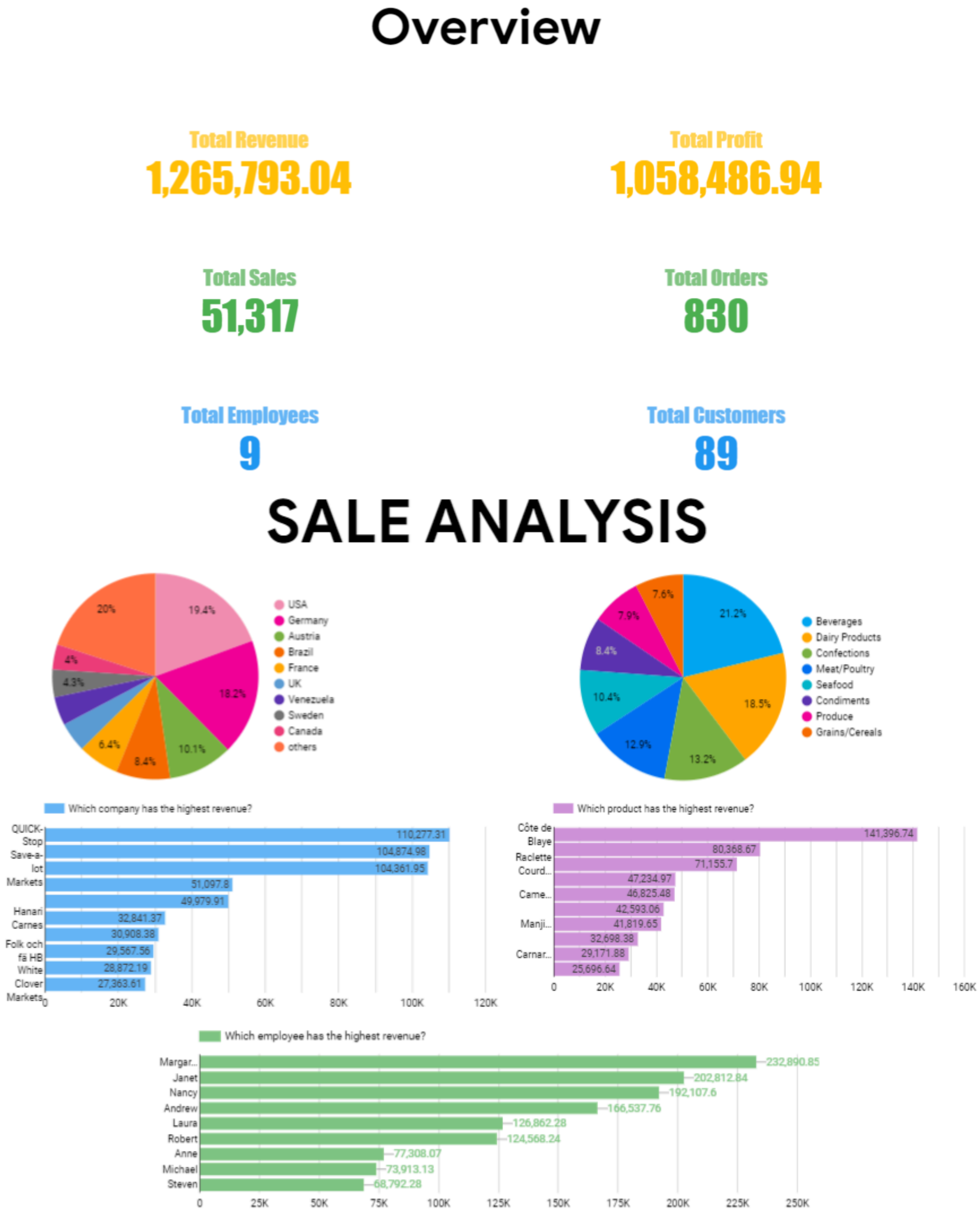
```sql
SELECT
  productID,
  productName,
  supplierID,
  categoryID,
  quantityPerUnit,
  unitPrice,
  unitsInStock,
  unitsOnOrder,
  reorderLevel,
  discontinued,
  current_timestamp() as  insertion_timestamp,
FROM
  `datawarehouse-411117.OLTP.products`
QUALIFY ROW_NUMBER() OVER(PARTITION BY productID) = 1;
```

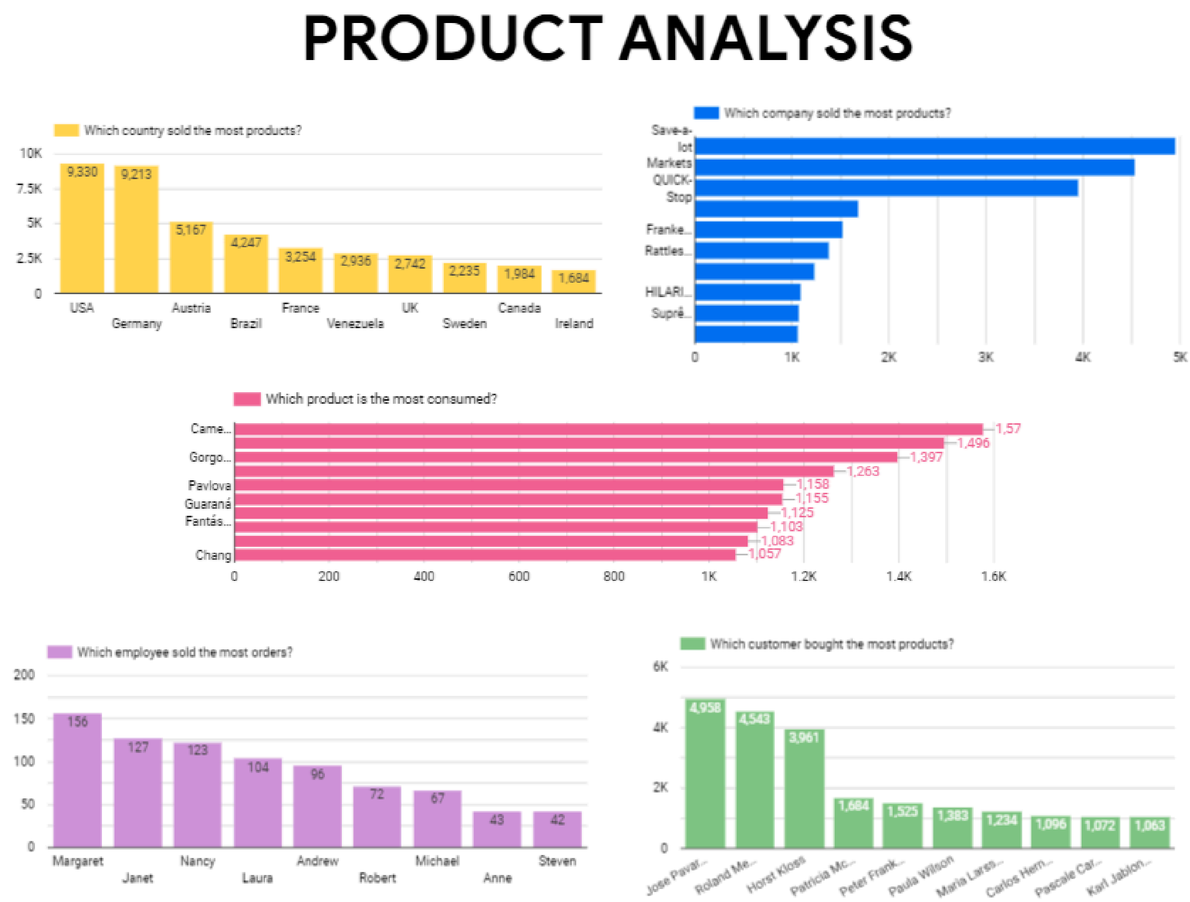
- dags\sql\fact_sales.sql:

```sql
CREATE OR REPLACE TABLE datawarehouse-411117.OLAP.Fact_sales AS
SELECT
  od.orderID,
  od.productID,
  o.customerID,
  o.employeeID,
  p.categoryID,
  od.quantity,
  od.unitPrice,
  od.discount,
  CAST(FORMAT_DATE('%Y%m%d', o.orderDate) AS INT64) AS date_key,
  o.freight,
  CURRENT_TIMESTAMP() AS insertion_timestamp
FROM
  `datawarehouse-411117.OLTP.order-details` od
LEFT JOIN `datawarehouse-411117.OLTP.orders` o
ON od.orderID = o.orderID
LEFT JOIN `datawarehouse-411117.OLTP.products` p
ON od.productID= p.productID
WHERE od.orderID IS NOT NULL
QUALIFY ROW_NUMBER() OVER(PARTITION BY o.orderID, od.productID) = 1;
```

V. **Visualization**

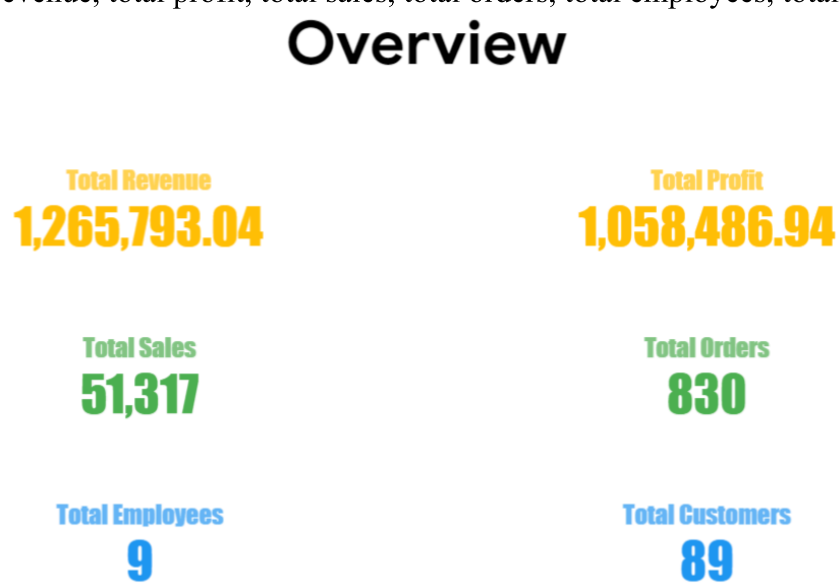
1. **Dashboard**



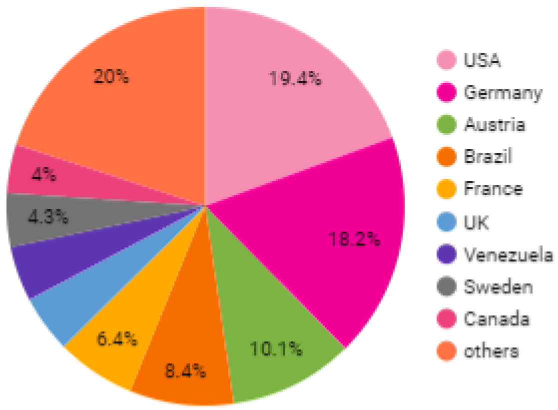


2. Answer business question

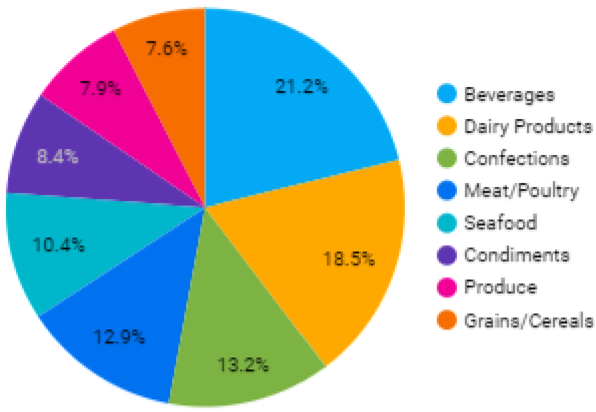
- Total revenue, total profit, total sales, total orders, total employees, total customers



- Which country has the highest revenue?

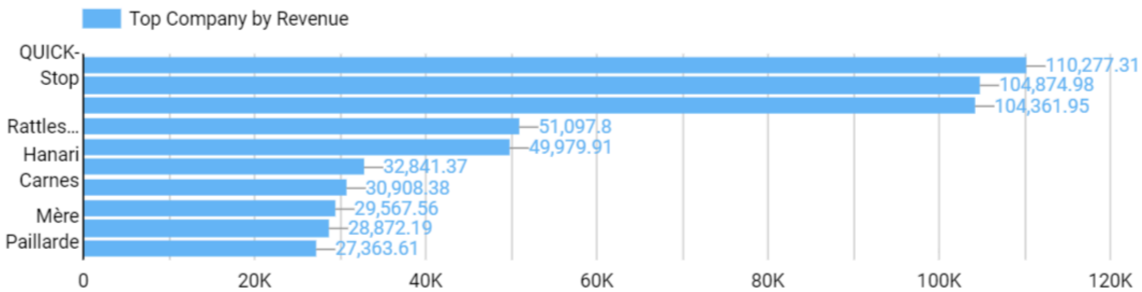


- Which product category has the highest sales?



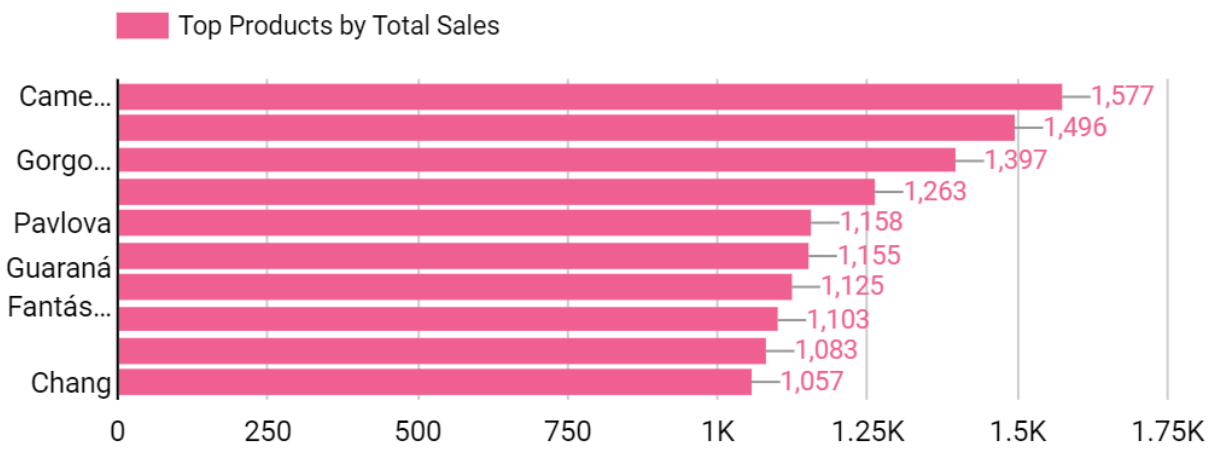
Beverages has the most revenue with 21,2%

- Which company has the highest total revenue?



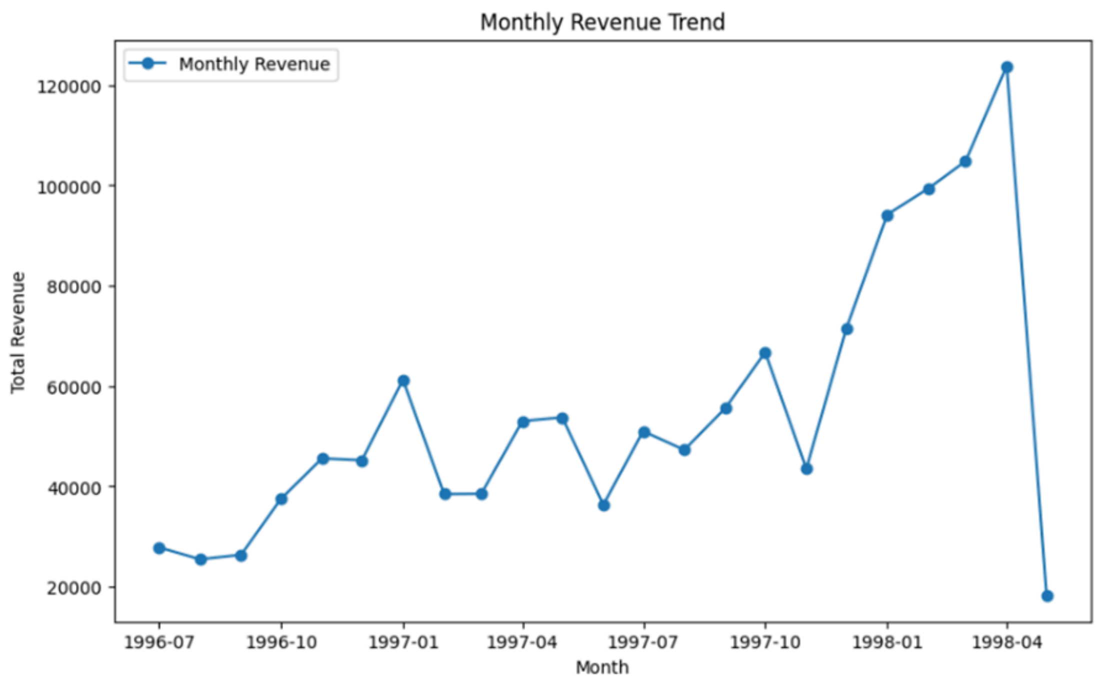
QUICK-Stop has the most revenue with $110,277

- Which product has the highest sales?

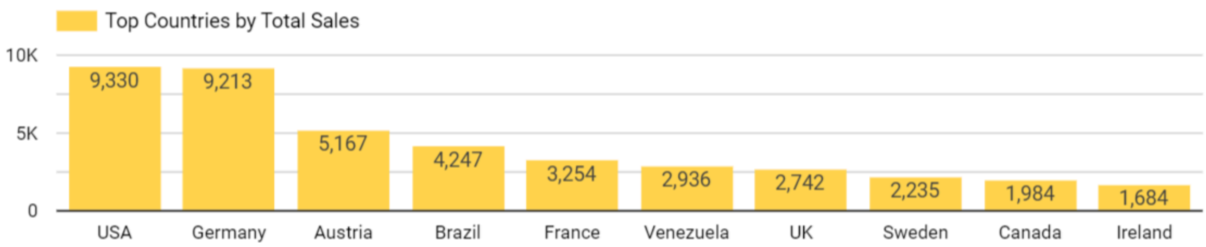


Camembert Pierrot has the most total sales with 1,577 products

- How does revenue change over the months of the year?

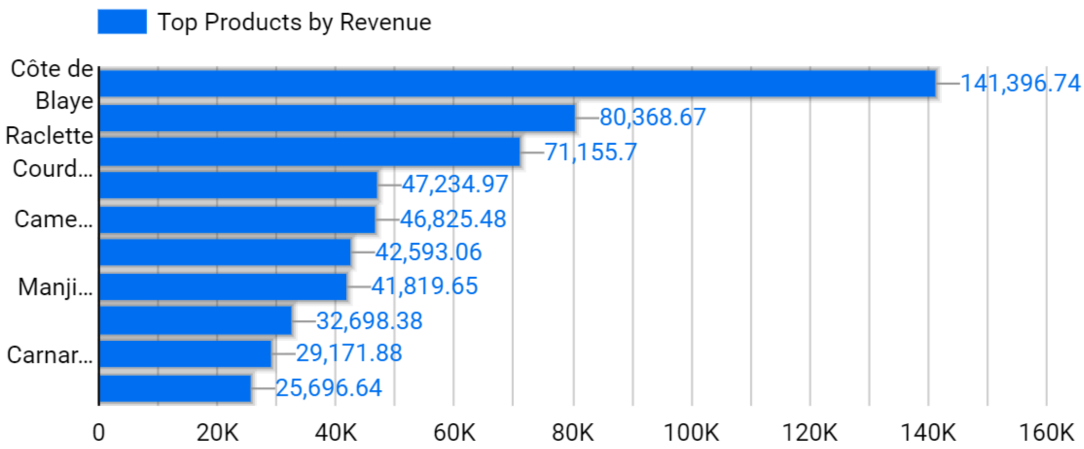


- Which products are sold the most, where they are sold?



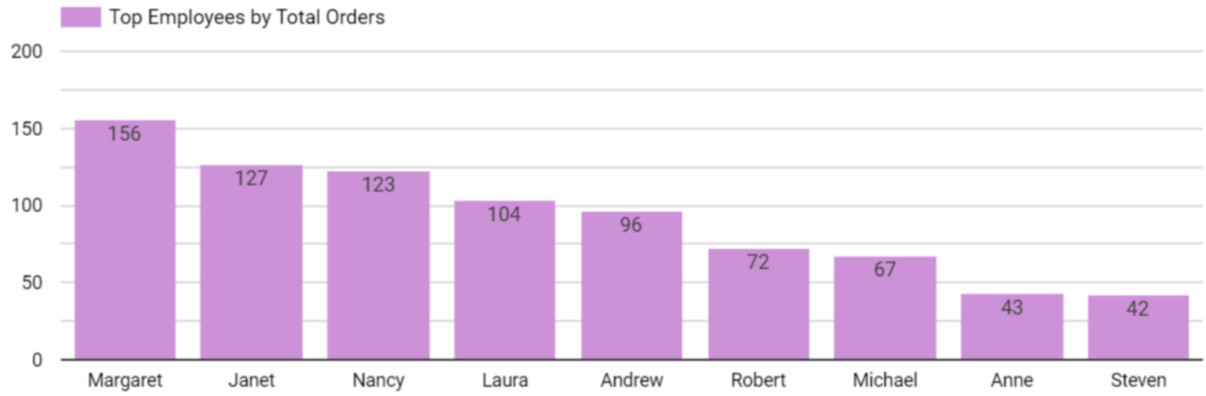
USA has the most product sold with 9,330 products

- Which products have the most Revenue?



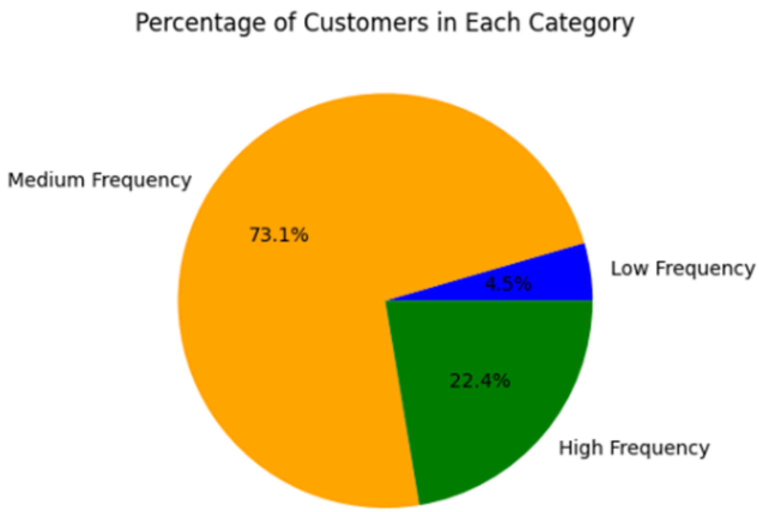
Côte de Blaye has the most revenue with $141,396

- Which employees have the most total sales?

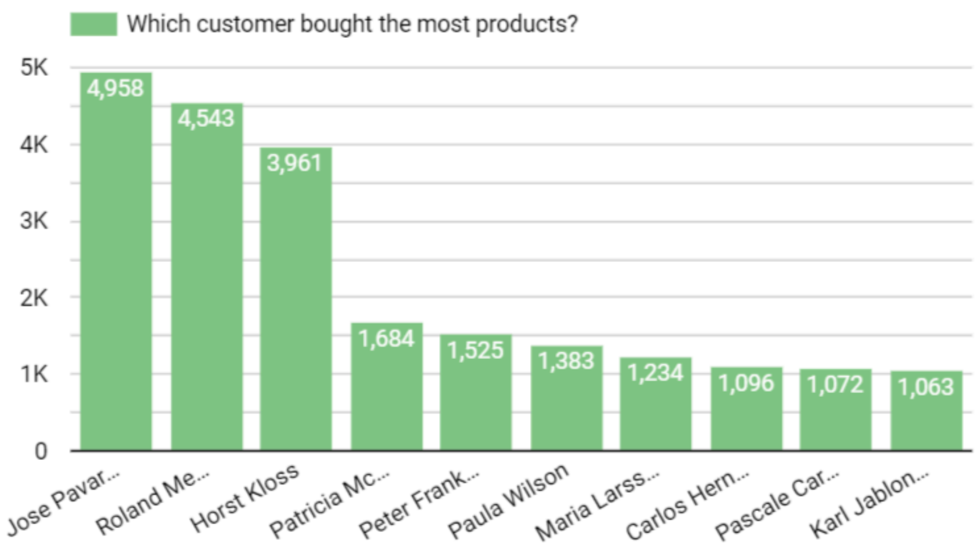


Margaret has the most total sales with 156 ordered

- Classify customers based on order frequency



- Which customer bought the most products?



Jose Pavarotti has bought the most products with 4,958

VI. Reference

1. Barnes, G. (n.d.). Implementing a Dimensional Data Warehouse with the SAS@System. [online]
   Available at:
   https://support.sas.com/resources/papers/proceedings/proceedings/sugi22/DATAWAREE/PAPER129.PDF [Accessed 17 Jan. 2024].
2. Simanjuntak, H., Nainggolan, A., Simatupang, D. and Manurung, D.D.V. (2017). An Automatic Tool to Transform Star Schema Data Warehouse to Physical Data Model. Journal of Telecommunication, Electronic and Computer Engineering (JTEC), [online] 9(2-3), pp.55–59.
   Available at: https://jtec.utem.edu.my/jtec/article/view/2273 .
3. Kimball Group. (n.d.). Data Warehouse Lifecycle Toolkit. [online]
   Available at:
   https://www.kimballgroup.com/data-warehouse-business-intelligence-resources/books/data-warehouse-dw-lifecycle-toolkit/ .