Assignment DEP

Saturday, 8 March 2025 10:45

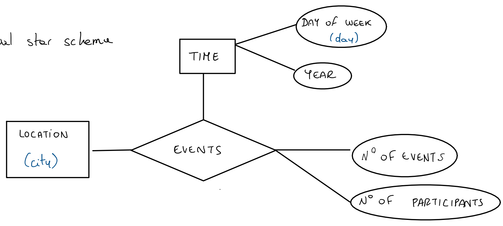
Assignment 1

1.1 a) The facts are 'Number of Events' and 'Total participants' — FACTS: numerical measure to be analyzed

The dimensions are the 'dayofweek', 'year' and 'location' — DIMENSIONS: descriptive data

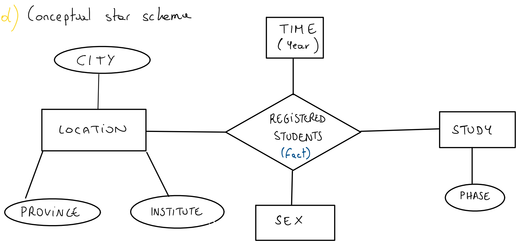
(CAN BE GROUPED)

b) Conceptual star schema



c) The facts are the number of registered students 'nrofregistrations'
The dimensions are: 'institute', 'city', 'province', 'study', 'phase', 'year', 'sex', 'instID', 'cityID'

d) Conceptual star schema



Assignment 4

a) For the 'goodness score' formula, one could consider some important aspects, such as:

- [H] n° of minutes spent on app : (timestamp of start game app - timestamp of end game app)
- [C] n° of completed games
- [T] open app times : n° of start of game app · day
- [B] boredom : switch to other apps · day

$$\text{GOODNESS} = \left(\frac{M}{\text{minutes in a day}} \cdot 100\right) + \left(2 \cdot C\right) + T - \underbrace{4}_{\text{penalizing factor}} \cdot B$$

a high value may indicate that the beta tester may enjoy spending time on the application and playing / completing a lot of games

b) Here we look for multiple business questions.

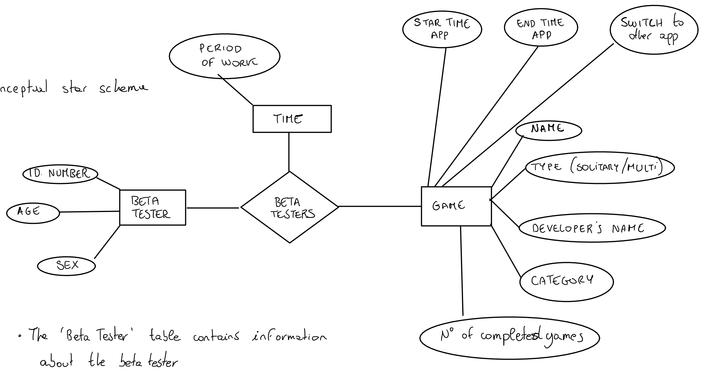
1) how to find the best beta player?
→ Should Pear choose him/her between streamers/Youtubers, or also between common people who enjoy the games genre? How many testers needed? From which country(s)? How can Pear know that the tester's feedbacks are trustworthy?

2) how to measure the effectiveness of its service?
→ Which data can Pear collect during this process? Should Pear share all the data to the developers? How to deal with the privacy of the testers? Which KPIs should be considered?

3) How can Pear monetize its service?
→ Which is the best pricing strategy? How much should the developers pay for the subscription, and how much for hiring the tester? More testers = More expenses? Or make a "tester group" discounted price? Will loyal customers (developers) receive discounts in subsequent years?

4) How can Pear optimize/improve its service?
→ Which data can be used to find better tester-developer matches? Should Pear take care of developers feedbacks before/after the service? Should Pear use external companies to evaluate the service level, or an internal team?

c) Conceptual star schema



- The 'Beta Tester' table contains information about the beta tester
  - ID_Number : Primary Key
  - Age, Sex as dimensions
- The 'Time' table contains information about the starting and, if applicable the end of working date of the beta tester
  - ID_Number : Foreign Key
  - Start date
  - End date
- The 'Game' table contains all the relevant information about the app/game
  - IDgame (primary Key)
  - Type, DevName, Category, GameName as dimensions Start/End Game
  - N° of completed games, Switch to other apps as fact
  - ID_Number : Foreign Key

1.3.3 Assignment 2: A simple ETL process to start with

Our data file for this assignment, is a spreadsheet file BI Raw Data.csv. The encoding of this file is “ISO- 8859-1”.3 A large warehouse located in the US has many customers for its different product from around the world. The warehouse manager Mr Jack Bezos has many large customers from over the world ordering different products from his warehouse. We follow the method of Section ?? where all steps are more or less already given.

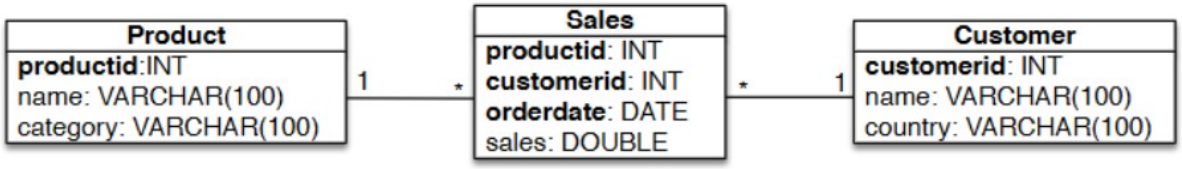


Figure 1.3: Starschema for Mr Bezos’ warehouse. Primary key in **bold** (if more attributes are in bold, they are *together* a primary key).

Step 1: Business questions (given)

Mr. Bezos wants to know answers for the following questions

- Who are his top-5 most valued customers?
- What are his top-5 most important products?

Step 2: Multidimensional model (given)

```
import pandas as pd
import numpy as np
from sqlalchemy import create_engine

data0=pd.read_csv('./BI_Raw_Data.csv',sep=';',encoding="ISO-8859-1")
# read first five rows of data
data0.head(5)
```

```
   Order_ID  Order_Date_Year  Order_Date_Month Order_Date_Day  \
0     11074             2009                 9      10-9-2009
1     11075             2009                 9      10-9-2009
2     11075             2009                 9      10-9-2009
3     11075             2009                 9      10-9-2009
4     11076             2009                 9      10-9-2009

   Order_Price_Total       Customer_Name Customer_Country  \
0              244.3       Simons bistro         Denmark
1              586.0  Richter Supermarkt      Switzerland
2              586.0  Richter Supermarkt      Switzerland
3              586.0  Richter Supermarkt      Switzerland
4             1057.0            Bon app'           France

                   Product_Name Product_Category
Product_Order_Unit_Price  \
0                       Pavlova      Confections
```

```
17.45
1                          Chang        Beverages
19.00
2                  Lakkalikööri        Beverages
18.00
3                     Spegesild          Seafood
12.00
4  Grandma's Boysenberry Spread       Condiments
25.00
```

```
   Product_Order_Quantity  Product_Order_Price_Total
0                      14                      244.3
1                      10                      190.0
2                       2                       36.0
3                      30                      360.0
4                      20                      500.0
```

```
# top 5 most valued customers
grouped_customers = data0.groupby('Customer_Name')
['Product_Order_Price_Total'].sum()
top5 = grouped_customers.sort_values(ascending=False)[0:5]
top5
```

```
Customer_Name
QUICK-Stop                      117483.39
Save-a-lot Markets              115673.39
Ernst Handel                    113236.68
Hungry Owl All-Night Grocers     57317.39
Rattlesnake Canyon Grocery       52245.90
Name: Product_Order_Price_Total, dtype: float64
```

```
# top 5 most important products
grouped_products = data0.groupby('Product_Name')
['Product_Order_Price_Total'].sum()
top5_products = grouped_products.sort_values(ascending=False)[0:5]
top5_products
```

```
Product_Name
Côte de Blaye               149984.2
Thüringer Rostbratwurst      87736.4
Raclette Courdavault         76296.0
Camembert Pierrot            50286.0
Tarte au sucre               49827.9
Name: Product_Order_Price_Total, dtype: float64
```

Step 3: Data exploration --> BI_RAW_DATA_EXCEL.xlsx

Step 4a: Create tables in your database

```
import psycopg2
```

```
# - Username and DB name:
# dab_ds2425-2a_10
# - Password:
# FYeq6/bll9ZFREsS
# - Example command for connecting:
# psql -h bronto.ewi.utwente.nl -U dab_ds2425-2a_10 dab_ds2425-2a_10
#- https://bronto.ewi.utwente.nl/phppgadmin/


DB_HOST = "bronto.ewi.utwente.nl"
DB_NAME = "dab_ds2425-2a_10"
DB_USER = "dab_ds2425-2a_10"
DB_PASS = "FYeq6/bll9ZFREsS"
DB_PORT = "5432"                # Default PostgreSQL port

# SQL queries to create tables
CREATE_TABLES_SQL = """
    CREATE TABLE IF NOT EXISTS ass2.customer (
        customerid SERIAL PRIMARY KEY,
        name VARCHAR(100),
        country VARCHAR(100)
    );

    CREATE TABLE IF NOT EXISTS ass2.product (
        productid SERIAL PRIMARY KEY,
        name VARCHAR(100),
        category VARCHAR(100)
    );

    CREATE TABLE IF NOT EXISTS ass2.sales (
        orderdate DATE NOT NULL,
        customerid INTEGER NOT NULL,
        productid INTEGER NOT NULL,
        sales DOUBLE PRECISION,
        PRIMARY KEY (orderdate, customerid, productid)
    );
"""

try:
    # Connect to PostgreSQL
    conn = psycopg2.connect(
        dbname=DB_NAME, user=DB_USER, password=DB_PASS, host=DB_HOST,
port=DB_PORT
    )
    conn.autocommit = True  # Ensure changes are saved automatically

    # Create a cursor object
    cursor = conn.cursor()

    # Execute table creation queries
```

```
    cursor.execute(CREATE_TABLES_SQL)

    print("Tables created successfully.")

    # Close the cursor and connection
    cursor.close()
    conn.close()

except Exception as e:
    print("Error:", e)
```

```
Tables created successfully.
```

Step 4b: ETL — Prepare data and fill the database

```
##data0=pd.read_csv('./BI_Raw_Data.csv',sep=';',encoding="ISO-8859-1")
data0.head()

# Step-1: Select the columns related to Product such Product_Name and
Product_Category
product=data0[['Product_Name', 'Product_Category']]
print(product.shape)
# Step 2: Rename the columns according to your star schema
product=product.rename(columns={'Product_Name':'name','Product_Categor
y':'category'})
# Step 3: Remove duplicatess.
# Note ignore_index will work only for Pandas version 1.0.0 or higher.
#If it gives error do not use it or update pandas version.
product=product.drop_duplicates(ignore_index=True,keep='last')
# Step 4: Generate an identifier for the dimension
# Resetting index to be sure it starts from zero.
product['productid'] = product.reset_index().index
# Step 5: re order the columns
product=product[['productid','name','category']]
# Print the shape of data((3,77) should be printed)
print(product.shape)
#Inspect first five rows( just a sanity check)
product.head()
```

```
(2155, 2)
(77, 3)
```

	productid	name	category
0	0	Mishi Kobe Niku	Meat/Poultry
1	1	Sirop d'érable	Condiments
2	2	Chocolade	Confections
3	3	Røgede sild	Seafood
4	4	Valkoinen suklaa	Confections

```
customers=data0[['Customer_Name', 'Customer_Country']]
print(customers.shape)
```

```
customers=customers.rename(columns={'Customer_Name':'name','Customer_C
ountry':'country'})
customers=customers.drop_duplicates(ignore_index=True,keep='last')
customers['customerid'] = customers.reset_index().index
customers=customers[['customerid','name','country']]
print(customers.shape)
customers.head()
```

```
(2155, 2)
(89, 3)
```

	customerid	name	country
0	0	La corne d'abondance	France
1	1	Spécialités du monde	France
2	2	France restauration	France
3	3	Alfreds Futterkiste	Germany
4	4	The Cracker Box	USA

```
#data0.columns
```

```
#data0.columns
sales0 = data0[['Order_Date_Day', 'Customer_Name', 'Product_Name',
'Product_Order_Price_Total']]
sales = sales0.merge(customers, left_on='Customer_Name',
right_on='name', how='left')
sales_final = sales.merge(product, left_on='Product_Name',
right_on='name', how='left')
sales_final['Order_Date_Day'] =
pd.to_datetime(sales_final['Order_Date_Day'], errors='coerce').dt.date
sales_final = sales_final[['Order_Date_Day', 'customerid',
'productid', 'Product_Order_Price_Total']]
sales_final = sales_final.rename(columns={'Order_Date_Day':
'orderdate', 'Product_Order_Price_Total': 'sales'})

# Handling missing values
sales_final = sales_final.dropna(subset=['customerid', 'productid'])

# Convert IDs to integers
sales_final['customerid'] = sales_final['customerid'].astype(int)
sales_final['productid'] = sales_final['productid'].astype(int)
print(sales_final.head())
```

	orderdate	customerid	productid	sales
0	2009-10-09	37	56	244.3
1	2009-10-09	82	55	190.0
2	2009-10-09	82	38	36.0
3	2009-10-09	82	23	360.0
4	2009-10-09	40	14	500.0

```
engine = create_engine(f'postgresql://{DB_NAME}:{DB_PASS}@{DB_HOST}:
{DB_PORT}/{DB_NAME}')
```

```
product.to_sql('product', engine, schema='ass2', index=False,
if_exists='replace')
customers.to_sql('customer', engine, schema='ass2', index=False,
if_exists='replace')
sales_final.to_sql('sales', engine, schema='ass2', index=False,
if_exists='replace')
```

```
155
```

Step 4: Visualize

```
import pandas as pd
from sqlalchemy import create_engine

DB_HOST = "bronto.ewi.utwente.nl"
DB_NAME = "dab_ds2425-2a_10"
DB_USER = "dab_ds2425-2a_10"
DB_PASS = "FYeq6/bll9ZFREsS"
DB_PORT = "5432"

engine = create_engine(f'postgresql://{DB_USER}:{DB_PASS}@{DB_HOST}:
{DB_PORT}/{DB_NAME}')

product = pd.read_sql('SELECT * FROM ass2.product', engine)
customers = pd.read_sql('SELECT * FROM ass2.customer', engine)
sales = pd.read_sql('SELECT * FROM ass2.sales', engine)

print(product.head())
print(customers.head())
print(sales.head())
```

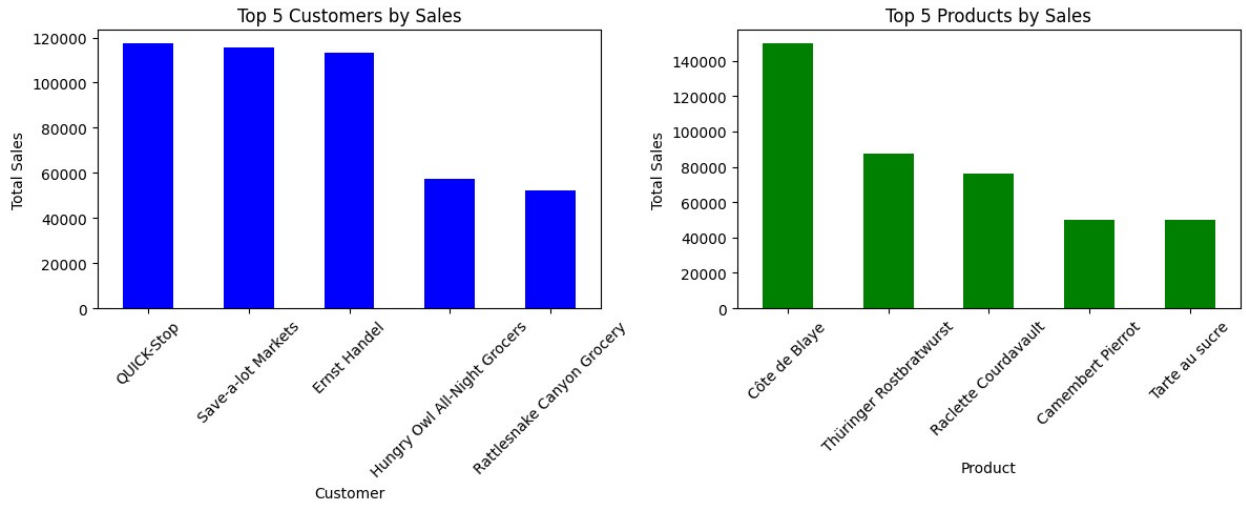
```
   productid              name      category
0          0   Mishi Kobe Niku  Meat/Poultry
1          1    Sirop d'érable    Condiments
2          2         Chocolade   Confections
3          3       Røgede sild       Seafood
4          4  Valkoinen suklaa   Confections
   customerid                    name  country
0           0  La corne d'abondance   France
1           1  Spécialités du monde   France
2           2   France restauration   France
3           3   Alfreds Futterkiste  Germany
4           4       The Cracker Box      USA
    orderdate  customerid  productid  sales
0  2009-10-09          37         56  244.3
1  2009-10-09          82         55  190.0
2  2009-10-09          82         38   36.0
3  2009-10-09          82         23  360.0
4  2009-10-09          40         14  500.0
```

NOTES: 1) The Orders table acts as a Fact Table (containing the facts: sales). The Customers and Products tables are Dimension Tables, providing context to the orders.

2) The Fact Table (Orders) is at the center, connected to the dimensions through keys (customerid, productid). The dimensions provide additional information about customers and products.

3) Potential Analysis: Sales by Product Category: - Summing sales for each category. - Sales by Country: Linking customerid to the customers table. - Temporal Trend: Analyzing sales trends over time...

```
import matplotlib.pyplot as plt

#sales_customers['Customer_Name'].value_counts().plt(kind='bar',
figsize=(20,5))

sales_customers= sales.merge(customers, on='customerid', how='left')
sales_products = sales.merge(product, on='productid', how='left')
top_customers = sales_customers.groupby('name')
['sales'].sum().nlargest(5)
top_products = sales_products.groupby('name')
['sales'].sum().nlargest(5)

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
top_customers.plot(kind='bar', color='blue')
plt.title('Top 5 Customers by Sales')
plt.xlabel('Customer')
plt.ylabel('Total Sales')
plt.xticks(rotation=45)

plt.subplot(1, 2, 2)
top_products.plot(kind='bar', color='green')
plt.title('Top 5 Products by Sales')
plt.xlabel('Product')
plt.ylabel('Total Sales')
plt.xticks(rotation=45)

plt.tight_layout()
plt.show()
```



NOTES: 1) The top three customers contribute heavily to total sales, suggesting they are high-priority clients.

2) The product Côte de Blaye is a top-seller, far exceeding the performance of other products.

3) Marketing efforts could be targeted towards the top customers to further increase sales or to other customers to balance revenue distribution.

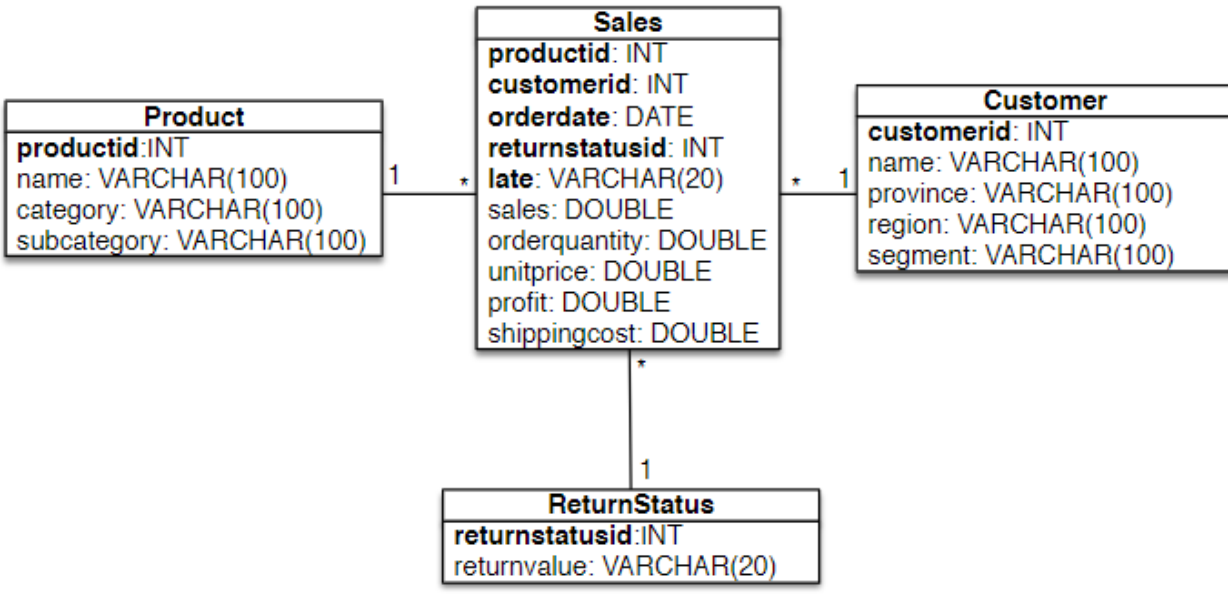


Figure 1.4: Starschema for SuperSales (logical design)

Step 1: Business questions (given)

For this new warehouse in Canada, Mr. Bezos wants to know answers for the following questions

- Which products/product categories made the most loss?
- Which products/product categories were shipped really late (more than 2 days)?
- Which products/product categories were returned the most?

```
import zipfile

zip_file_path = "./SuperSales.zip"
extract_to = "./"

with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall(extract_to)

print("Unzipping complete!")
```

```
Unzipping complete!
```

```
import os
import pandas as pd

for file in os.listdir('./SuperSales'):
    print(file)
```

```
main =
pd.read_csv('./SuperSales/SuperstoreSales_main.csv',sep=';',encoding="
ISO-8859-1")
manager =
pd.read_csv('./SuperSales/SuperstoreSales_manager.csv',sep=';',encodin
g="ISO-8859-1")
returns =
pd.read_csv('./SuperSales/SuperstoreSales_returns.csv',sep=';',encodin
g="ISO-8859-1")
```

```
SuperstoreSales_main.csv
SuperstoreSales_manager.csv
SuperstoreSales_returns.csv
```

```
print("main:",main.columns)
print("manager:",manager.columns)
print("ret:",returns.columns)
```

```
main: Index(['Row ID', 'Order ID', 'Order Date', 'Order Priority',
'Order Quantity',
       'Sales', 'Discount', 'Ship Mode', 'Profit', 'Unit Price',
       'Shipping Cost', 'Customer Name', 'Province', 'Region',
       'Customer Segment', 'Product Category', 'Product Sub-Category',
       'Product Name', 'Product Container', 'Product Base Margin',
       'Ship Date'],
      dtype='object')
manager: Index(['Region', 'Manager'], dtype='object')
ret: Index(['Order ID', 'Status'], dtype='object')
```

```
# top 5 loss
## Profit = Sales - (Cost + Discounts + Shipping, etc.)
main_returned = pd.merge(returns,main,how='left',on='Order ID')

### replacing to float
main_returned['Profit'] =
main_returned.Profit.astype(str).str.replace(',','.').astype(float)
top5_loss_products = main_returned.groupby('Product Name')
['Profit'].sum().sort_values(ascending=True)[0:5]
top5_loss_products_cat = main_returned.groupby('Product Category')
['Profit'].sum().sort_values(ascending=True)[0:5]
print(f"""Top 5 Products that made the most loss: \n
{top5_loss_products},
      \n ----------------------------- \n
      Top 5 Products Categories that made the most loss: \n
{top5_loss_products_cat}""")
```

```
Top 5 Products that made the most loss:
 Product Name
Polycom ViewStationª ISDN Videoconferencing Unit                -
```

```
11984.40
Riverside Palais Royal Lawyers Bookcase, Royale Cherry Finish   -
11053.60
Polycom ViewStationª Adapter H323 Videoconferencing Unit        -
4858.67
Hon iLevelª Computer Training Table                             -
4283.90
Bush Mission Pointe Library                                     -
3837.46
Name: Profit, dtype: float64,

 ----------------------------

      Top 5 Products Categories that made the most loss:
 Product Category
Furniture           -3487.53
Office Supplies    100921.47
Technology         100998.96
Name: Profit, dtype: float64
```

```python
def convert_two_digit_year(date_str):
    # Convert two-digit years (e.g., '10' -> '2010')
    parts = date_str.split('/')
    if len(parts[2]) == 2:
        parts[2] = '20' + parts[2]  # Assumes years in the 2000s
    return '/'.join(parts)

main['Order Date'] = main['Order Date'].apply(convert_two_digit_year)
main['Ship Date'] = main['Ship Date'].apply(convert_two_digit_year)

# Now convert to datetime
main['Order Date'] = pd.to_datetime(main['Order Date'],
format='%d/%m/%Y')
main['Ship Date'] = pd.to_datetime(main['Ship Date'],
format='%d/%m/%Y')

# Calculate Delivery Days
main['Delivery Days'] = (main['Ship Date'] - main['Order
Date']).dt.days

main['Order Date'] = pd.to_datetime(main['Order Date'],
format='%d/%m/%Y', errors='coerce')
main['Ship Date'] = pd.to_datetime(main['Ship Date'],
format='%d/%m/%Y', errors='coerce')

# If needed, you can check if there are any invalid dates:
invalid_orders = main[main['Order Date'].isna()]
invalid_ships = main[main['Ship Date'].isna()]

main['Order Date'] = pd.to_datetime(main['Order Date'],
```

```
infer_datetime_format=True, errors='coerce')
main['Ship Date'] = pd.to_datetime(main['Ship Date'],
infer_datetime_format=True, errors='coerce')

# Calculate Delivery Days
main['Delivery Days'] = (main['Ship Date'] - main['Order
Date']).dt.days
```

```
C:\Users\Mardeen\AppData\Local\Temp\ipykernel_9288\3059251151.py:25:
UserWarning: The argument 'infer_datetime_format' is deprecated and
will be removed in a future version. A strict version of it is now the
default, see https://pandas.pydata.org/pdeps/0004-consistent-to-
datetime-parsing.html. You can safely remove this argument.
  main['Order Date'] = pd.to_datetime(main['Order Date'],
infer_datetime_format=True, errors='coerce')
C:\Users\Mardeen\AppData\Local\Temp\ipykernel_9288\3059251151.py:26:
UserWarning: The argument 'infer_datetime_format' is deprecated and
will be removed in a future version. A strict version of it is now the
default, see https://pandas.pydata.org/pdeps/0004-consistent-to-
datetime-parsing.html. You can safely remove this argument.
  main['Ship Date'] = pd.to_datetime(main['Ship Date'],
infer_datetime_format=True, errors='coerce')
```

```
# Filter only late shipments (Delivery Days > 2)
late_shipments = main[main['Delivery Days'] > 2]
# Count occurrences of late shipments per product
top5_late_products = late_shipments.groupby('Product
Name').size().sort_values(ascending=False).head(5)
# Count occurrences of late shipments per product category
top5_late_products_cat = late_shipments.groupby('Product
Category').size().sort_values(ascending=False).head(5)
print(f"""Top 5 Products that were shipped really late (more than 2
days): \n {top5_late_products},
      \n ----------------------------- \n
      Top 5 Product Categories that were shipped really late (more
than 2 days): \n {top5_late_products_cat}""")
```

```
Top 5 Products that were shipped really late (more than 2 days):
 Product Name
Global Deluxe Stacking Chair, Gray                    7
Microsoft Internet Keyboard                           6
Global High-Back Leather Tilter, Burgundy             6
Bush Advantage Collection¨ Round Conference Table     6
Avery 506                                             6
dtype: int64,

 -----------------------------

      Top 5 Product Categories that were shipped really late (more
than 2 days):
```

```
 Product Category
Office Supplies    840
Technology         375
Furniture          318
dtype: int64
```

```
top5_returned_products = main_returned.groupby('Product
Name').size().sort_values(ascending=False).head(5)
top5_returned_products_cat = main_returned.groupby('Product
Category').size().sort_values(ascending=False).head(5)
print(f"""Top 5 Products that were most returned: \n
{top5_returned_products},
      \n ---------------------------- \n
      Top 5 Products Categories that were most returned: \n
{top5_returned_products_cat}""")
```

```
Top 5 Products that were most returned:
 Product Name
O'Sullivan Elevations Bookcase, Cherry Finish      5
Bush Mission Pointe Library                        5
Imation Neon Mac Format Diskettes, 10/Pack         5
Coloredge Poster Frame                             5
Riverside Furniture Stanwyck Manor Table Series    5
dtype: int64,

 ----------------------------

      Top 5 Products Categories that were most returned:
 Product Category
Office Supplies    461
Technology         218
Furniture          193
dtype: int64
```

Step 2: Multidimensional model (given)

ReturnStatus Table (Normalized Approach)

```
import psycopg2
from sqlalchemy import create_engine

DB_HOST = "bronto.ewi.utwente.nl"
DB_NAME = "dab_ds2425-2a_10"
DB_USER = "dab_ds2425-2a_10"
DB_PASS = "FYeq6/bll9ZFREsS"
DB_PORT = "5432"                # Default PostgreSQL port
engine = create_engine(f'postgresql://{DB_NAME}:{DB_PASS}@{DB_HOST}:
{DB_PORT}/{DB_NAME}')

CREATE_TABLES_SQL = """
```

```
    CREATE SCHEMA IF NOT EXISTS ass3;

    CREATE TABLE IF NOT EXISTS ass3.product (
        productId INT PRIMARY KEY,
        name VARCHAR(100),
        category VARCHAR(100),
        subcategory VARCHAR(100)
    );

    CREATE TABLE IF NOT EXISTS ass3.customer (
        customerId INT PRIMARY KEY,
        name VARCHAR(100),
        province VARCHAR(100),
        region VARCHAR(100),
        segment VARCHAR(100)
    );

    CREATE TABLE IF NOT EXISTS ass3.returnstatus (
        returnstatusId INT PRIMARY KEY,
        returnvalue VARCHAR(20)
    );

    CREATE TABLE IF NOT EXISTS ass3.sales (
        productId INT,
        customerId INT,
        orderdate DATE,
        returnstatusId INT,
        late VARCHAR(20) CHECK (late IN ('Late', 'NotLate')),
        sales DOUBLE PRECISION,
        orderquantity DOUBLE PRECISION,
        unitprice DOUBLE PRECISION,
        profit DOUBLE PRECISION,
        shippingcost DOUBLE PRECISION,
        FOREIGN KEY (productId) REFERENCES ass3.product(productId),
        FOREIGN KEY (customerId) REFERENCES ass3.customer(customerId),
        FOREIGN KEY (returnstatusId) REFERENCES
ass3.returnstatus(returnstatusId)
    );
"""

try:
    conn = psycopg2.connect(
        dbname=DB_NAME, user=DB_USER, password=DB_PASS, host=DB_HOST,
port=DB_PORT
    )
    conn.autocommit = True
    cursor = conn.cursor()
    cursor.execute(CREATE_TABLES_SQL)
    print("Tables created successfully.")
    cursor.close()
```

```
    conn.close()
except Exception as e:
    print("Error:", e)
```

```
Tables created successfully.
```

Step 3: Data exploration: Excel

Step 4: ETL — Prepare data and fill the database

Grouping and Aggregating the data

```
print("main:",main.columns)
print("manager:",manager.columns)
print("ret:",returns.columns)

##Return Status
return_status = pd.DataFrame({
    'returnstatusid': [0, 1],
    'returnvalue': ['NotReturned', 'Returned']
})

return_status.to_sql('returnstatus', engine, schema='ass3',
if_exists='replace', index=False)

## Product
product_df = main[['Product Name', 'Product Category', 'Product Sub-
Category']].drop_duplicates().reset_index(drop=True)
product_df['productId'] = product_df.index
product_df = product_df.rename(columns={
    'Product Name': 'name',
    'Product Category': 'category',
    'Product Sub-Category': 'subcategory'
})
product_df.to_sql('product', engine, schema='ass3',
if_exists='replace', index=False)

## Customer
customer_df = main[['Customer Name', 'Province', 'Region', 'Customer
Segment']].drop_duplicates().reset_index(drop=True)
customer_df['customerId'] = customer_df.index
customer_df = customer_df.rename(columns={
    'Customer Name': 'name',
    'Customer Segment': 'segment'
})
customer_df.to_sql('customer', engine, schema='ass3',
if_exists='replace', index=False)
```

```
main: Index(['Row ID', 'Order ID', 'Order Date', 'Order Priority',
'Order Quantity',
```

```
       'Sales', 'Discount', 'Ship Mode', 'Profit', 'Unit Price',
       'Shipping Cost', 'Customer Name', 'Province', 'Region',
       'Customer Segment', 'Product Category', 'Product Sub-Category',
       'Product Name', 'Product Container', 'Product Base Margin',
'Ship Date',
       'Delivery Days'],
      dtype='object')
manager: Index(['Region', 'Manager'], dtype='object')
ret: Index(['Order ID', 'Status'], dtype='object')

832

# Merge main and return table for sales
main_merged = main.merge(returns, on='Order ID', how='left')  # ret
contains 'Order ID', 'Status'
main_merged['Status'] = main_merged['Status'].fillna('NotReturned')  #
fill missing returns
main_merged['returnstatusId'] =
main_merged['Status'].map({'NotReturned': 0, 'Returned': 1})

# Add "Late" column (inlined dimension)
main_merged['late'] = main_merged['Delivery Days'].apply(lambda x:
'Late' if x > 2 else 'NotLate')

# Map productId and customerId from earlier dataframes
main_merged = main_merged.merge(product_df[['productId', 'name']],
left_on='Product Name', right_on='name')
main_merged = main_merged.merge(customer_df[['customerId', 'name']],
left_on='Customer Name', right_on='name', suffixes=('_prod', '_cust'))

# Prepare Sales fact table
sales_df = main_merged[[
    'productId', 'customerId', 'Order Date', 'returnstatusId', 'late',
    'Sales', 'Order Quantity', 'Unit Price', 'Profit', 'Shipping Cost'
]].rename(columns={
    'Order Date': 'orderdate',
    'Sales': 'sales',
    'Order Quantity': 'orderquantity',
    'Unit Price': 'unitprice',
    'Profit': 'profit',
    'Shipping Cost': 'shippingcost'
})

# Convert date column
sales_df['orderdate'] = pd.to_datetime(sales_df['orderdate'])

# Write to sales table
#sales_df.to_sql('sales', engine, schema='ass3', if_exists='replace',
index=False)
#print("Data loaded into PostgreSQL successfully.")
```

```
cols_to_convert1 = ['sales','unitprice','profit','shippingcost']
for col in cols_to_convert1:
    sales_df[col] = sales_df[col].str.replace(',', '.', regex=False)
# replace comma with dot

# Group and aggregate
# Convert numeric columns (in case they were read as strings)
cols_to_convert = ['sales', 'orderquantity', 'unitprice', 'profit',
'shippingcost']
for col in cols_to_convert:
    sales_df[col] = pd.to_numeric(sales_df[col], errors='coerce')  #
invalid strings will become NaN

sales_grouped = sales_df.groupby(
    ['productId', 'customerId', 'orderdate', 'late',
'returnstatusId'],
    as_index=False
).agg({
    'sales': 'sum',
    'orderquantity': 'sum',
    'unitprice': 'mean',         # or first(), if it's always the same
per group
    'profit': 'sum',
    'shippingcost': 'sum'
})

# Optional: round values
sales_grouped['unitprice'] = sales_grouped['unitprice'].round(2)

sales_grouped
# Load into the database
sales_grouped.to_sql('sales', engine, schema='ass3',
if_exists='replace', index=False)
```

```
353
```

Step 4: Visualization

```
## See tables
from sqlalchemy import text
with engine.connect() as conn:
    result = conn.execute(text("""SELECT table_name
                           FROM information_schema.tables
                           WHERE table_schema='ass3'""""))
    print(result.fetchall())

pd.read_sql_table('product', engine,schema='ass3').info()
pd.read_sql_table('customer', engine,schema='ass3').info()
pd.read_sql_table('sales', engine,schema='ass3').info()
pd.read_sql_table('returnstatus', engine,schema='ass3').info()
```

```
[('sales',), ('returnstatus',), ('product',), ('customer',)]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1263 entries, 0 to 1262
Data columns (total 4 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   name         1263 non-null   object
 1   category     1263 non-null   object
 2   subcategory  1263 non-null   object
 3   productId    1263 non-null   int64 
dtypes: int64(1), object(3)
memory usage: 39.6+ KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1832 entries, 0 to 1831
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   name        1832 non-null   object
 1   Province    1832 non-null   object
 2   Region      1832 non-null   object
 3   segment     1832 non-null   object
 4   customerId  1832 non-null   int64 
dtypes: int64(1), object(4)
memory usage: 71.7+ KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23353 entries, 0 to 23352
Data columns (total 10 columns):
 #   Column          Non-Null Count  Dtype         
---  ------          --------------  -----         
 0   productId       23353 non-null  int64         
 1   customerId      23353 non-null  int64         
 2   orderdate       23353 non-null  datetime64[ns]
 3   late            23353 non-null  object        
 4   returnstatusId  23353 non-null  int64         
 5   sales           23353 non-null  float64       
 6   orderquantity   23353 non-null  int64         
 7   unitprice       23353 non-null  float64       
 8   profit          23353 non-null  float64       
 9   shippingcost    23353 non-null  float64       
dtypes: datetime64[ns](1), float64(4), int64(4), object(1)
memory usage: 1.8+ MB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2 entries, 0 to 1
Data columns (total 2 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   returnstatusid  2 non-null      int64 
 1   returnvalue     2 non-null      object
```

```
dtypes: int64(1), object(1)
memory usage: 164.0+ bytes
```

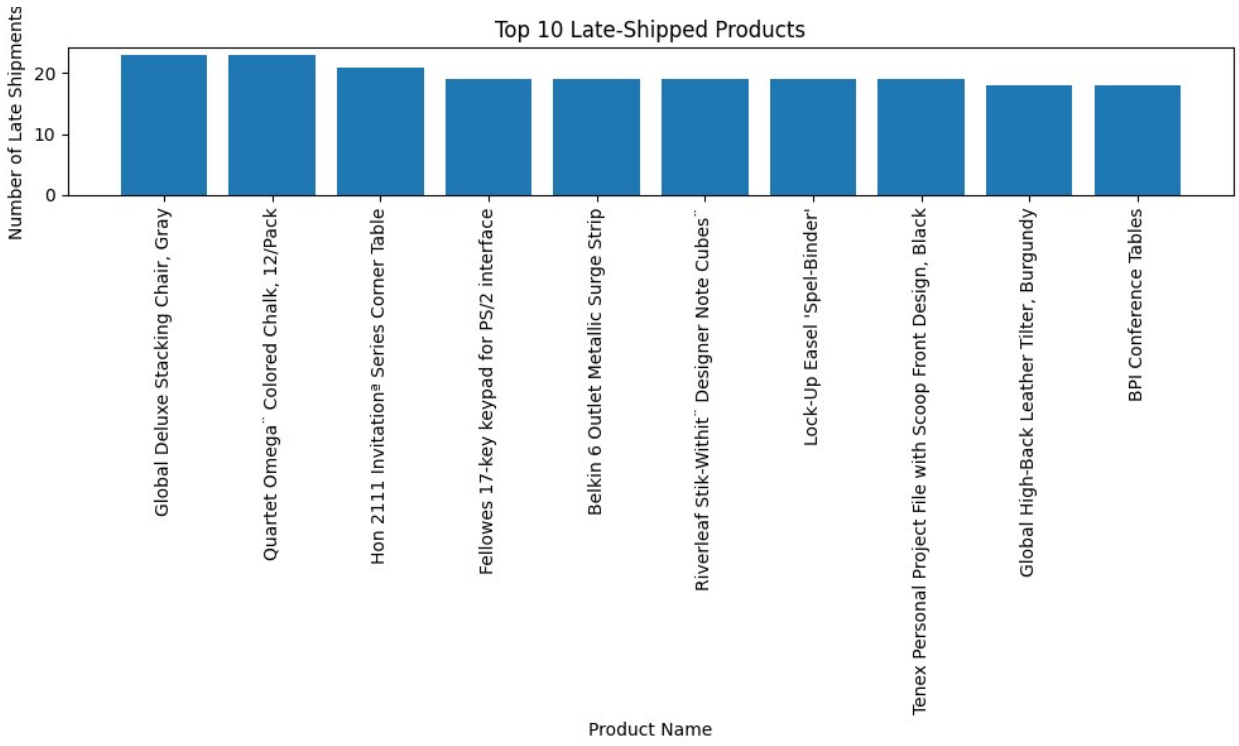
Which products/product categories were shipped really late (more than 2 days)?

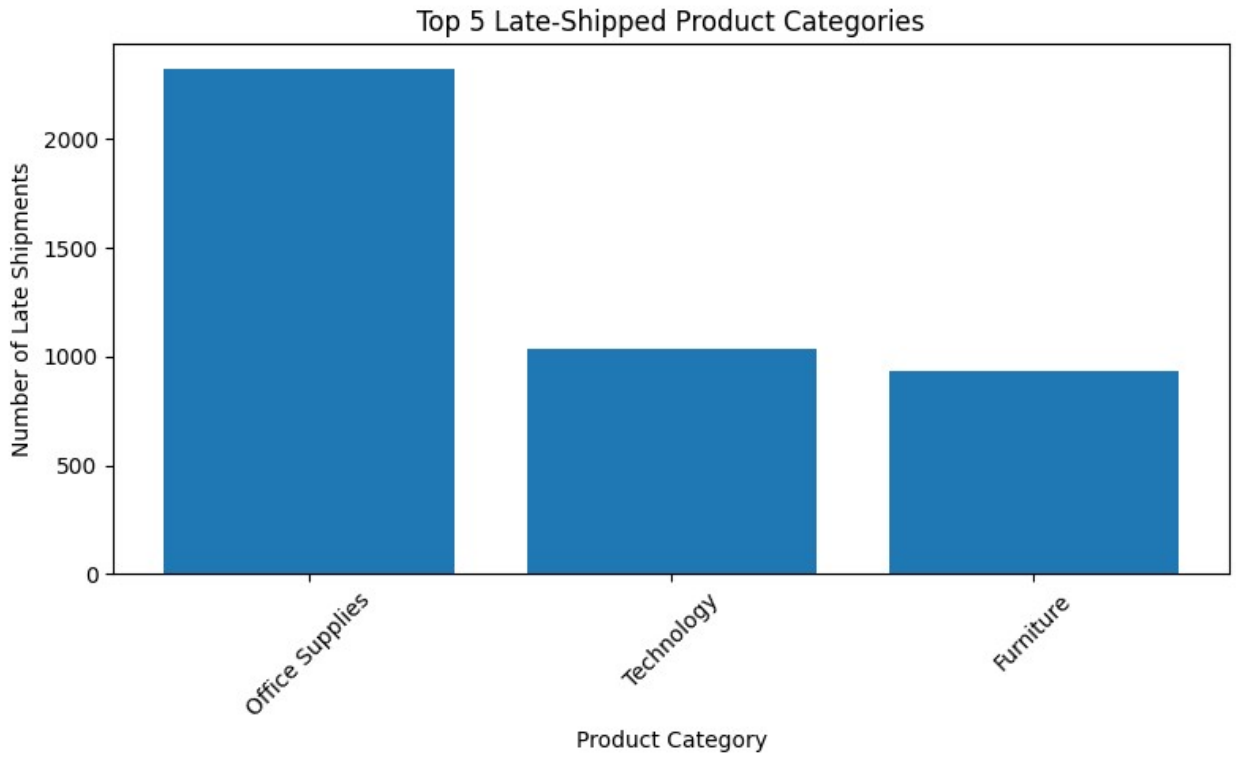
```
from sqlalchemy import text

query = text("""
    SELECT p."name" AS product_name,
           p."category" AS product_category,
           s."shippingcost",
           s."orderdate",
           s."customerId"
    FROM ass3.sales s
    JOIN ass3.product p ON s."productId" = p."productId"
    WHERE s."late" = 'Late'
""")

with engine.connect() as conn:
    df_late_products = pd.read_sql_query(query, conn)

df_late_products.head()
```

```
                                        product_name
product_category  \
0  Eldon Base for stackable storage shelf, platinum  Office Supplies

1  Eldon Base for stackable storage shelf, platinum  Office Supplies

2  Eldon Base for stackable storage shelf, platinum  Office Supplies

3  Cardinal Slant-D¨ Ring Binder, Heavy Gauge Vinyl  Office Supplies

4  Cardinal Slant-D¨ Ring Binder, Heavy Gauge Vinyl  Office Supplies


   shippingcost  orderdate  customerId
0         35.00 2010-10-13           0
1         35.00 2010-10-13          64
2         35.00 2010-10-13         118
3          2.99 2010-08-04         507
4          2.99 2012-11-11         635
```

```
import matplotlib.pyplot as plt

# 1. Top 10 late products by count
top_late_products =
df_late_products['product_name'].value_counts().head(10)
```

```
plt.figure(figsize=(10, 6))
plt.bar(top_late_products.index, top_late_products.values)
plt.xlabel('Product Name')
plt.ylabel('Number of Late Shipments')
plt.title('Top 10 Late-Shipped Products')
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()

# 2. Top 5 product categories by count
top_late_categories =
df_late_products['product_category'].value_counts().head(5)

plt.figure(figsize=(8, 5))
plt.bar(top_late_categories.index, top_late_categories.values)
plt.xlabel('Product Category')
plt.ylabel('Number of Late Shipments')
plt.title('Top 5 Late-Shipped Product Categories')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```




Top 5 Late-Shipped Product Categories
Number of Late Shipments
2000
1500
1000
500
0
Office Supplies
Technology
Furniture
Product Category

Step 3: Excel

One important thing to check here is the following. Two different files are relevant now: a main one with the orders, and one that contains the returned orders. Do not fail to check that these two files ‘ connect properly to each other: do all orders in the returned orders file match existing orders in the main file and vice versa? You can follow the following strategy to check this with Excel:

· Import both files into Excel as two different sheets: “Sheet1” and “Sheet2”. Insert a new column on the far left in both where we will put the results of our matching attempts.

· Sheet1: Sheet with returned orders: in cell A2 put the formula “=IF(ISNA(VLOOKUP(B2;Sheet2!C:C;1;FALSE));0;1)”. Adapt the formula to your situation: this one assumes the OrderID is in column B in Sheet1 and in column C in Sheet2. The formula puts a 1 if the OrderID is matched with an existing order in the main file, or a 0 if there is not a match. Copy the formula to all other cells in the A-column. You could put in cell A1 the formula “=COUNTIF(A2:An;0)” to quickly determine how many orders in this sheet are not matched. It should be zero.

· Sheet2: Sheet with main file with orders: We do the same in the other sheet: matching the OrderID in column C with the OrderID of Sheet1 in column B.

· You should see that in Sheet1 all the orders are matched: the number of unmatched orders is zero. You will also see that in Sheet2, not all orders are matched: most are not, but some are matched.

· Make a note how many returned orders are in the one file (they were all matched), how many orders were not returned, and how many orders are there in total (which obviously should add up).

	A	B	C	D	E
1	0	Order ID	Status		
2	1	65	Returned		
3	1	69	Returned		
4	1	134	Returned		
5	1	135	Returned		
6	1	230	Returned		
7	1	324	Returned		
8	1	359	Returned		
9	1	612	Returned		
10	1	614	Returned		
11	1	678	Returned		
12	1	710	Returned		
13	1	740	Returned		
14	1	775	Returned		
15	1	833	Returned		
16	1	902	Returned		
17	1	928	Returned		
18	1	930	Returned		
19	1	1060	Returned		
20	1	1127	Returned		
21	1	1285	Returned		
22	1	1317	Returned		
23	1	1382	Returned		
24	1	1538	Returned		
25	1	1665	Returned		

Foglio1 Sheet1 Sheet2

	A	B	C	D	E
1	7527	Row ID	Order ID	Order Date	Order P
2	0	1	3	13/10/2010	Low
3	0	49	293	01/10/2012	High
4	0	50	293	01/10/2012	High
5	0	80	483	10/07/2011	High
6	0	85	515	28/08/2010	Not Spe
7	0	86	515	28/08/2010	Not Spe
8	0	97	613	17/06/2011	High
9	0	98	613	17/06/2011	High
10	0	103	643	24/03/2011	High
11	1	107	678	26/02/2010	Low
12	0	127	807	23/11/2010	Medium
13	0	128	807	23/11/2010	Medium
14	0	134	868	08/06/2012	Not Spe
15	0	135	868	08/06/2012	Not Spe
16	0	149	933	04/08/2012	Not Spe
17	0	160	995	30/05/2011	Medium
18	0	161	998	25/11/2009	Not Spe
19	0	175	1154	14/02/2012	Critical
20	0	176	1154	14/02/2012	Critical
21	0	203	1344	15/04/2012	Low
22	0	204	1344	15/04/2012	Low
23	0	213	1412	12/03/2010	Not Spe
24	0	214	1412	12/03/2010	Not Spe
25	0	229	1539	09/03/2011	Low

Foglio1 Sheet1 Sheet2

Note:

572	orders matched and Returned	Sheet1
872	orders matched and Returned	Sheet2
7527	orders not returned	Sheet2
8399	total orders	Sheet2

And there is one other issue that we like to warn you about. The star schema models a proper cube in the pure sense of multidimensional modeling. For each combination of dimension values, there should be exactly one row in the fact table. Suppose that the same customer orders the same product twice on the same day and both products are not late and not returned. In other words, all dimensions are the same for these two orders. If this happens, we need to combine (aggregate) the facts of these two orders, so that we end up with exactly one row for this particular combination of dimension values.

So, does this happen in our data? We cannot see easily check this in the source data because we do not have the data prepared yet for lateness and returned or not, but let's do the next best thing: are there rows in the main file that have the same value for Order date, Customer name, and Product name? You can follow the following strategy to check this with Excel:

· Make a copy of Sheet2. Delete all columns except the three just mentioned.

· Sort it on all 4 columns.

· In cell D3, put a formula "=IF(AND(A3=A2;B3=B2;C3=C2);1;0)". It will produce 1 if all values of the current row are the same as the previous one. Copy the formula in the cell to all other rows.

· Search for a value of 1 in this column.

You will see that there are two such cases!

	A	B	C	D	E
1	Order Date	Customer Name	Product Name	2	
2	01/01/2009	Matt Collister	Safco Industrial Wire Shelving	0	
3	01/01/2009	Jessica Myrick	Perma STOR-ALL[a] Hanging File Box, 13 1/8"W x 12 1/4"D x 10 1/2"H	0	
4	02/01/2009	David Philippe	Avery Trapezoid Ring Binder, 3" Capacity, Black, 1040 sheets	0	
5	02/01/2009	Alan Schoenberger	Hon 4070 Series Pagoda[a] Armless Upholstered Stacking Chairs	0	
6	02/01/2009	Alan Schoenberger	Hon Valutask[a] Swivel Chairs	0	
7	02/01/2009	Alan Schoenberger	Dual Level, Single-Width Filing Carts	0	
8	02/01/2009	Elizabeth Moffitt	Black Print Carbonless Snap-Off¨ Rapid Letter, 8 1/2" x 7"	0	
9	02/01/2009	Elizabeth Moffitt	White GlueTop Scratch Pads	0	
10	03/01/2009	Bill Donatelli	Xerox 4200 Series MultiUse Premium Copy Paper (20Lb. and 84 Bright)	0	
11	03/01/2009	Bill Donatelli	T18	0	
164	21/01/2009	Andrew Allen	TOPS Money Receipt Book, Consecutively Numbered in Red,	0	
165	21/01/2009	Larry Tron	Hon 4700 Series Mobuis[a] Mid-Back Task Chairs with Adjustable Arms	0	
166	21/01/2009	Larry Tron	Southworth 25% Cotton Premium Laser Paper and Envelopes	0	
167	21/01/2009	Larry Tron	Southworth 25% Cotton Premium Laser Paper and Envelopes	1	
168	21/01/2009	Roy Phan	XtraLife¨ ClearVue[a] Slant-D¨ Ring Binders by Cardinal	0	
169	21/01/2009	Roy Phan	Deflect-o DuraMat Antistatic Studded Beveled Mat for Medium Pile Carpeting	0	
170	22/01/2009	Khloe Miller	Career Cubicle Clock, 8 1/4", Black	0	
4047	24/11/2010	Dave Hallsten	Staples Copy Paper (20Lb. and 84 Bright)	0	
4048	24/11/2010	Dave Hallsten	Col-Erase¨ Pencils with Erasers	0	
4049	24/11/2010	Ellis Ballard	600 Series Non-Flip	0	
4050	24/11/2010	Liz Price	Verbatim DVD-RAM, 9.4GB, Rewritable, Type 1, DS, DataLife Plus	0	
4051	24/11/2010	Liz Price	Verbatim DVD-RAM, 9.4GB, Rewritable, Type 1, DS, DataLife Plus	1	
4052	24/11/2010	Brenda Bowman	Belkin ErgoBoard[a] Keyboard	0	
4053	24/11/2010	Kristen Hastings	Accessory9	0	