FKlubCase

—

Tasks

1. **Task A -** Start the Virtual machine
2. **Task B -** Choose business process(es)
3. **Task C -** Dimensional modeling
4. **Task D -** Design and implement an ETL flow
5. **Task E -** Create your cube(s) in Mondrian
6. **Task F -** Create reports and analyze the data

Task B

Business process:

Fklub sells different products (drinks and snacks) to students and staff.

The data provided by the FKlub give us information about products sold, members buying products, point of sale...

The business process we are choosing to work on is the sales of products, as the sales process is the one with the largest potential for increasing the profits. For which reason this business process should thus be prioritized.

Granularity:

It is important to use the data granularity that best matches the analysis needs, since the grain has a huge influence on the precision of a fact.

For the business process “Sales of products”, the granularity “Total sales per product per member per month” may be detailed enough, while still enabling performance and storage gain.

Questions:

The questions concerning the sales of products that we will try to answer are :

1. Statistic sale per product per year
2. Top 10 products sold per month
3. Products sold the most per year
4. Total sales per month

Task C

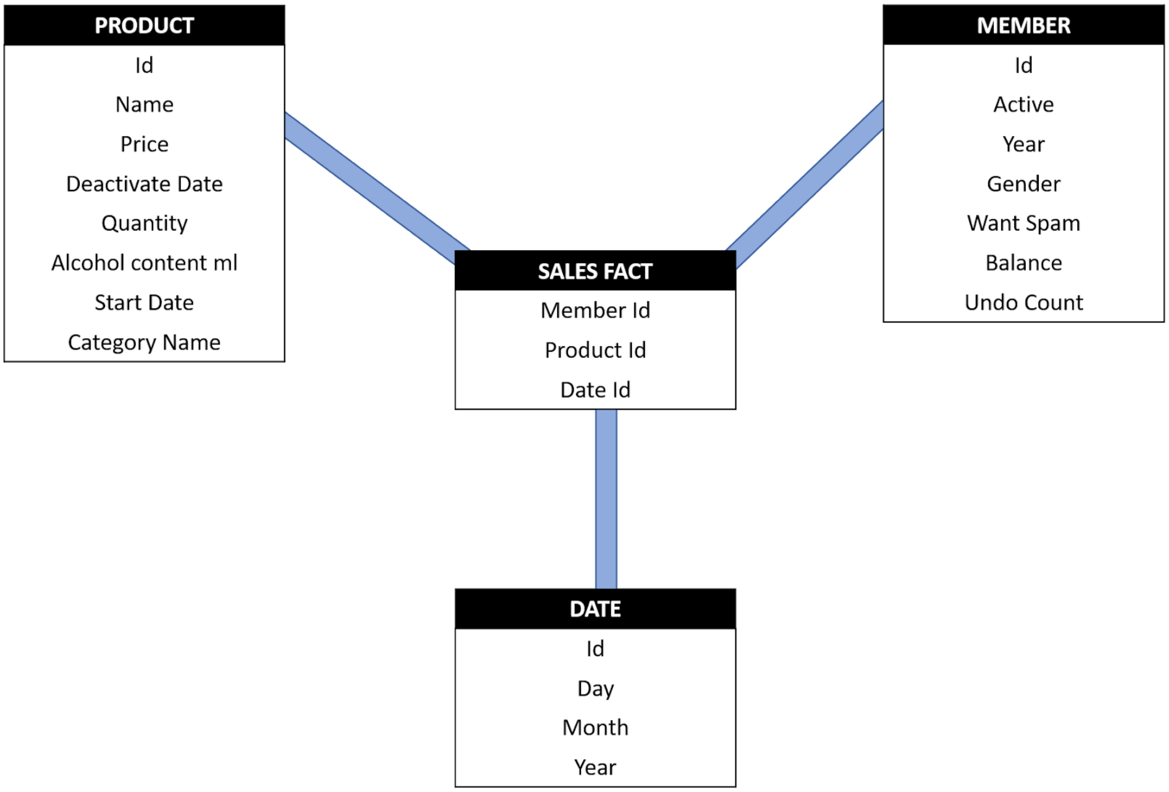


Figure 1: Star schema

Dimensions:

The dimensions that we decided to include to answer the “Sales of products” business process are Product, Member and Date.

Measures:

The measures chosen to help us answer this business process are Amount of product sold and Price of sales.

SCDs:

Slowly Changing Dimension is used when you wish to capture the changing data within the dimension over time. Therefore, having the Product Dimension as a SCD would be very beneficial for FKlub as they could easily analyse, for example, how the price change would affect the number of sales.

A type 2 update will be used as it allows to accurately keep all historical information and ensure optimal use of the active record.

Relational Schema

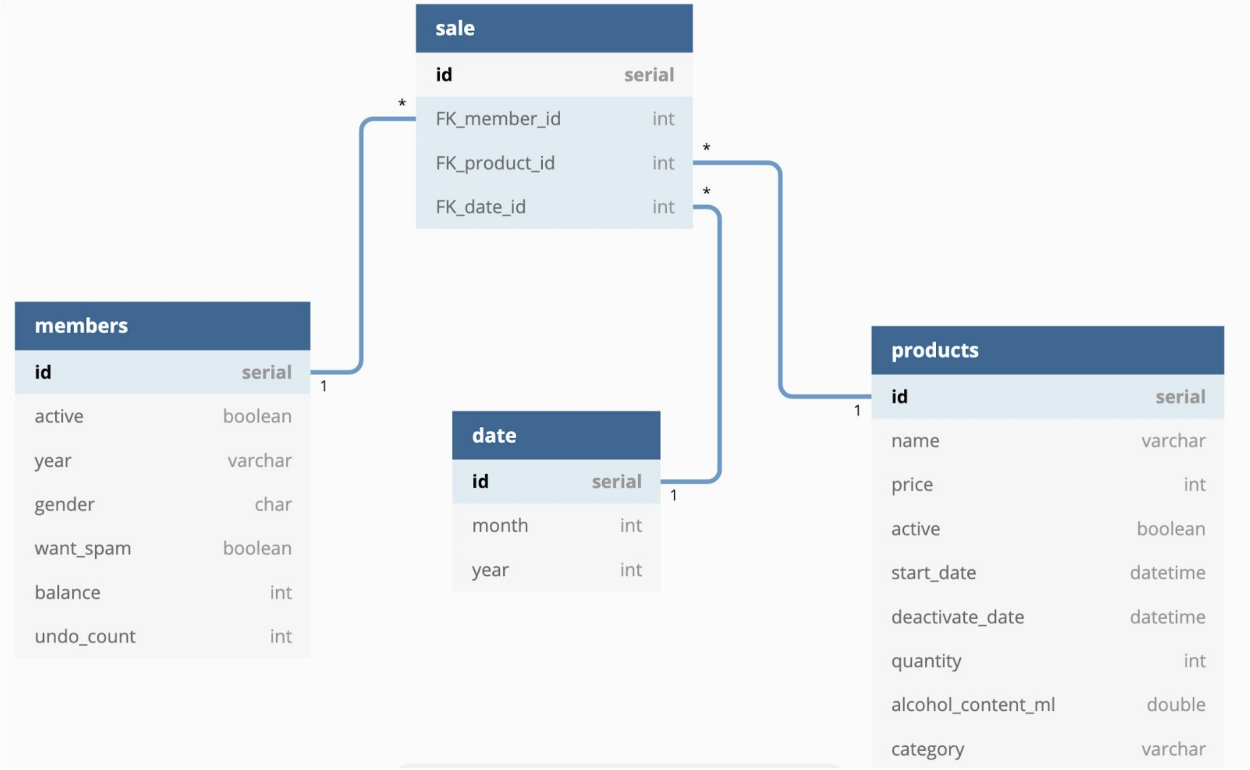
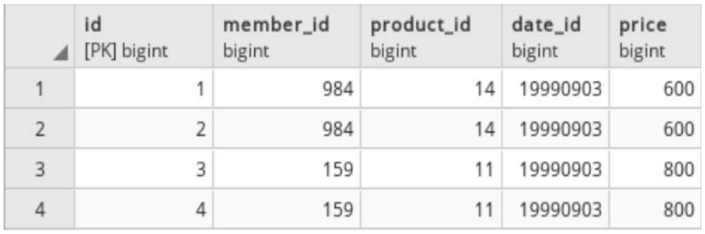


Figure 2 : Relational schema

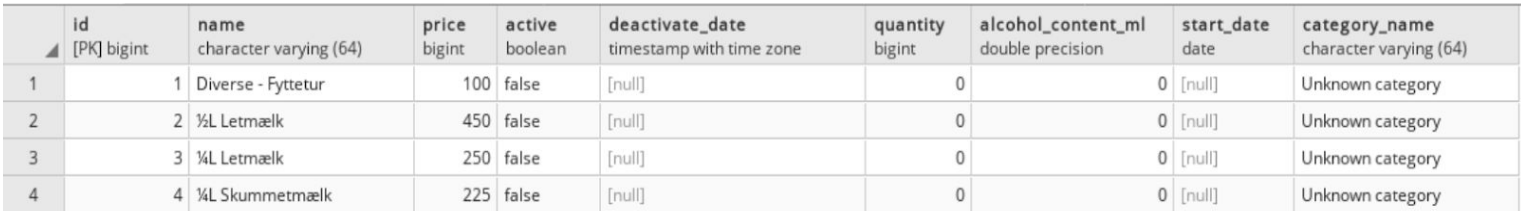
Relational Representation

Sale - Fact table



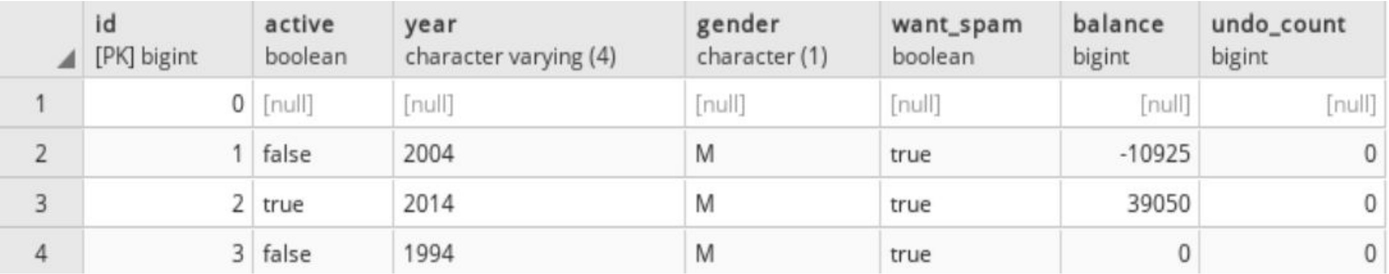
	id [PK] bigint	member_id bigint	product_id bigint	date_id bigint	price bigint
1	1	984	14	19990903	600
2	2	984	14	19990903	600
3	3	159	11	19990903	800
4	4	159	11	19990903	800

Product dimension



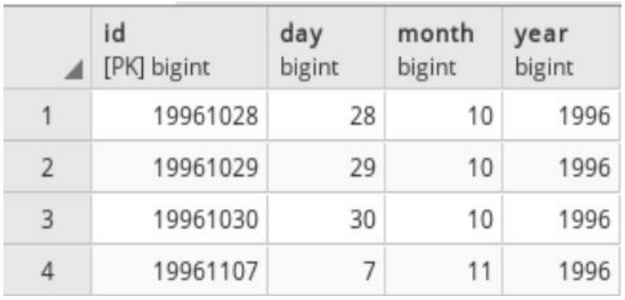
	id [PK] bigint	name character varying (64)	price bigint	active boolean	deactivate_date timestamp with time zone	quantity bigint	alcohol_content_ml double precision	start_date date	category_name character varying (64)
1	1	Diverse - Fyttetur	100	false	[null]	0	0	[null]	Unknown category
2	2	½L Letmælk	450	false	[null]	0	0	[null]	Unknown category
3	3	¼L Letmælk	250	false	[null]	0	0	[null]	Unknown category
4	4	¼L Skummetmælk	225	false	[null]	0	0	[null]	Unknown category

Member dimension



	id	active	year	gender	want_spam	balance	undo_count
	[PK] bigint	boolean	character varying (4)	character (1)	boolean	bigint	bigint
1	0	[null]	[null]	[null]	[null]	[null]	[null]
2	1	false	2004	M	true	-10925	0
3	2	true	2014	M	true	39050	0
4	3	false	1994	M	true	0	0

Date dimension



	id	day	month	year
	[PK] bigint	bigint	bigint	bigint
1	19961028	28	10	1996
2	19961029	29	10	1996
3	19961030	30	10	1996
4	19961107	7	11	1996

Task D

ETL Flow

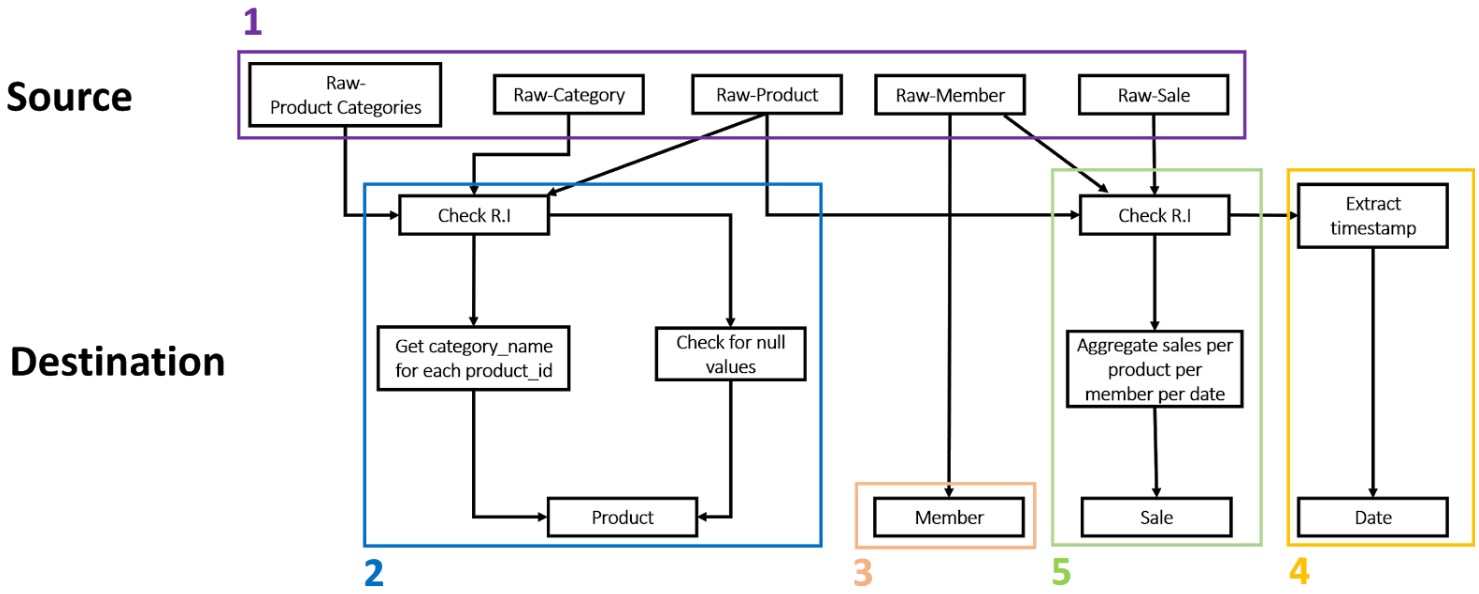


Figure 3 : ETL Flow

Figure 3 illustrates the initial load for creating a relational data warehouse which consists of 3 dimension tables and a fact table. Dimensions: Product, Member, Date whereas our fact table is represented by Sales table. In our solution, we have generated Date as our dimension table by converting the 'timestamp' feature found in sales.csv file into 3 different features: day, month, year.

1. Source files

Looking at the top of our ETL flow we can see Raw-Product Categories, Raw-Category, Raw-Product, Raw-Member and Raw-Sale which are represented in our case by the following csv files: product_categories.csv, category.csv, product.csv, member.csv and sale.csv files.

In order to perform the initial load, we first had to prepare a script in Python3 which would generate the aforementioned tables in PostgreSQL: product_dimension_table, member_dimension_table, date_dimension_table, and sales_fact_table .

2. Storing data in product_dimension_table

First, we obtained the category_name of each product. We would read data from category.csv in order to save the category names for each category id and then we would read from product_category.csv and save a dictionary with the category name corresponded to that product id.

To store data in product_dimension_table, we would read data from product.csv, using pygrametl CSVSource class and save data using list comprehension by storing data into required dimension objects. For each product, we would also store a row containing its category name.

3. Storing data in member_dimension_table

To store data in member_dimension_table we would read data from member.csv, using pygrametl CSVSource class and save data using list comprehension by storing data into required dimension objects.

4. Storing data in date_dimension_table

In order to store data regarding date_dimension_table we have created a method convert_timestamp() which takes as input the 'timestamp' feature from sales.csv file. This method would generate 3 different columns such as: year, month, day and return them for later use.

In our method load_data() we iterate over all records from sales.csv file where we pass the 'timestamp' as input to convert_timestamp() method and generate for each sale record 3 additional new features: year, month, day. These features would represent date_dimension_table columns whereas the date_id represents a unique smart key composed of year-month-day.

5. Storing data in sales_fact_table

To save data into sales_fact_table we would get the member_id, product_id, and the newly generated smart key for date_id and store data accordingly into each column.

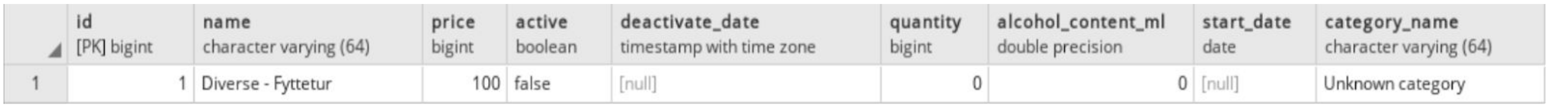
The 'Aggregate sales per product per member per date' module represents our process in creation of sales statistics for the Fklub.

Observations:

The following are some of the observations we have seen while performing ETL (Extract-Transform-Load) using pygrametl library:

(1) Empty values in Product Dimension

Perform check against null values for some of the columns in product.csv file (e.g category_name), then, replace these null values with special dimension values (e.g. ‘Unknown category).



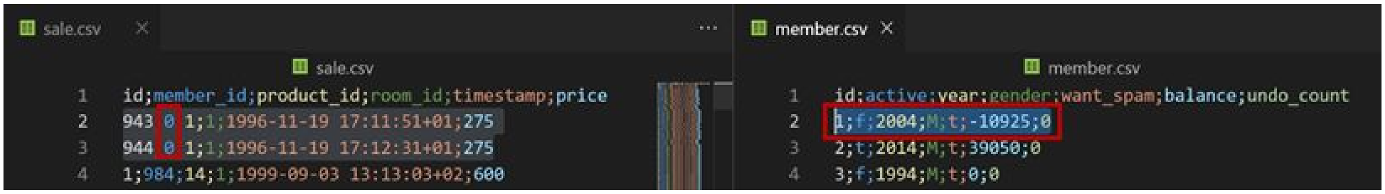
	id [PK] bigint	name character varying (64)	price bigint	active boolean	deactivate_date timestamp with time zone	quantity bigint	alcohol_content_ml double precision	start_date date	category_name character varying (64)
1	1	Diverse - Fyttetur	100	false	[null]	0	0	[null]	Unknown category

Figure 4 : Use of special dimension values

The reason is that the null values in a database must be given a systematic and uniform treatment so we can avoid problems in joins and be able to store consistent data in product_dimension_table. This is a very important rule because a null can be interpreted as data is missing, data is not known or data is not applicable.

(2) Inconsistency in the generated dump files (e.g member.csv, sale.csv files)

Extracting and loading member data from member.csv to member_dimension_table would result to an incomplete information. The reason is that while extracting and loading sales data from sale.csv to sales_fact_table we found missing data with respect to some of member_id(s).



```
id;member_id;product_id;room_id;timestamp;price
943 0 1;1;1996-11-19 17:11:51+01;275
944 0 1;1;1996-11-19 17:12:31+01;275
1;984;14;1;1999-09-03 13:13:03+02;600
```

```
id;active;year;gender;want_spam;balance;undo_count
1;f;2004;M;t;-10925;0
2;t;2014;M;t;39050;0
3;f;1994;M;t;0;0
```

Figure 5 : Inconsistency in the generated dump files

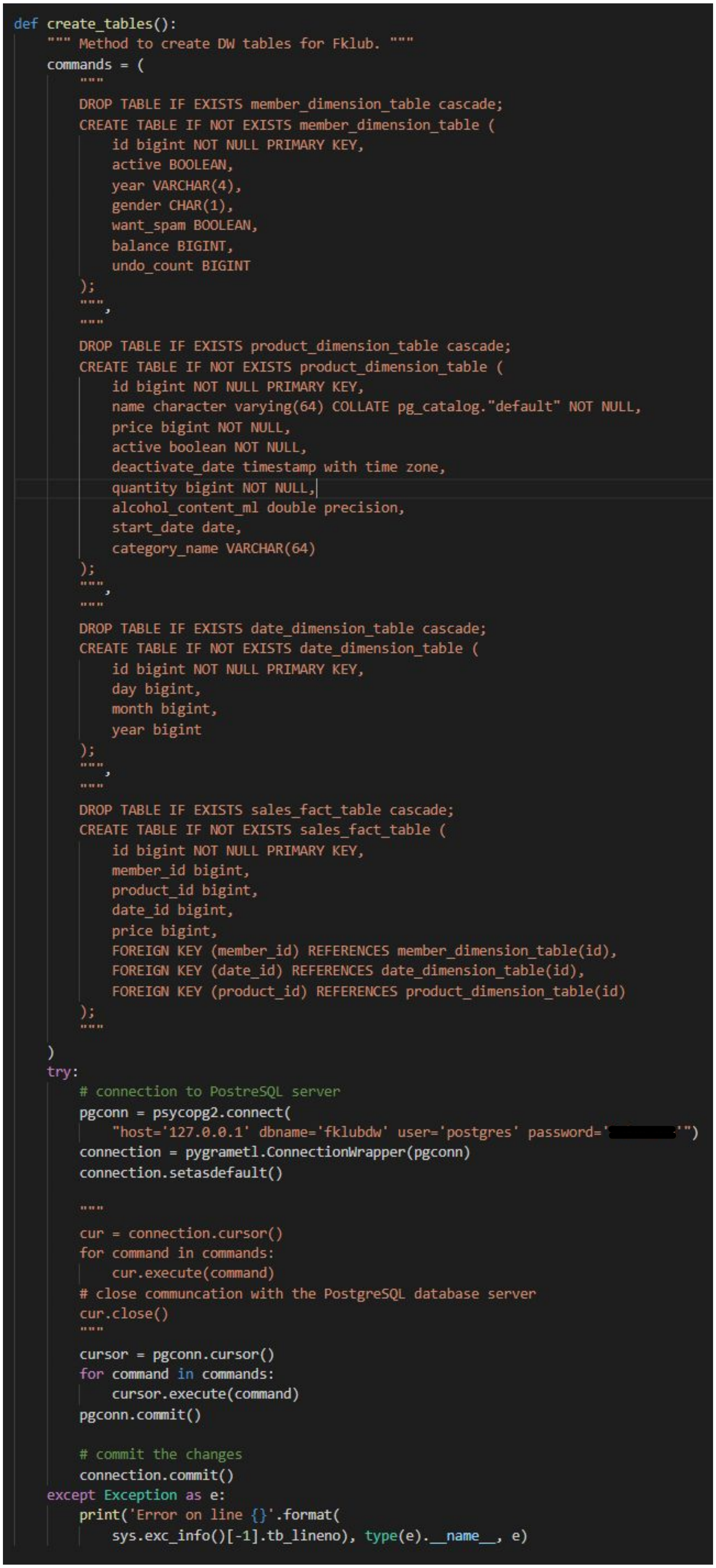
(*) member.csv file: all ids start from 1 as auto increment while in sales.csv file, member id starts the count from 0.

In our proposed solution we have created an SQL query that would gather all the member ids from member_dimension_table and store them into a list of member ids.

The given list is then used to perform a check such as taking each member_id from sales_fact_table and look up for existing member_id in the list. If the member_id is not found, then we would append it to the list of member_ids and saved to it into member_dimension_table using SQL INSERT statement operation, resulting in a complete data.

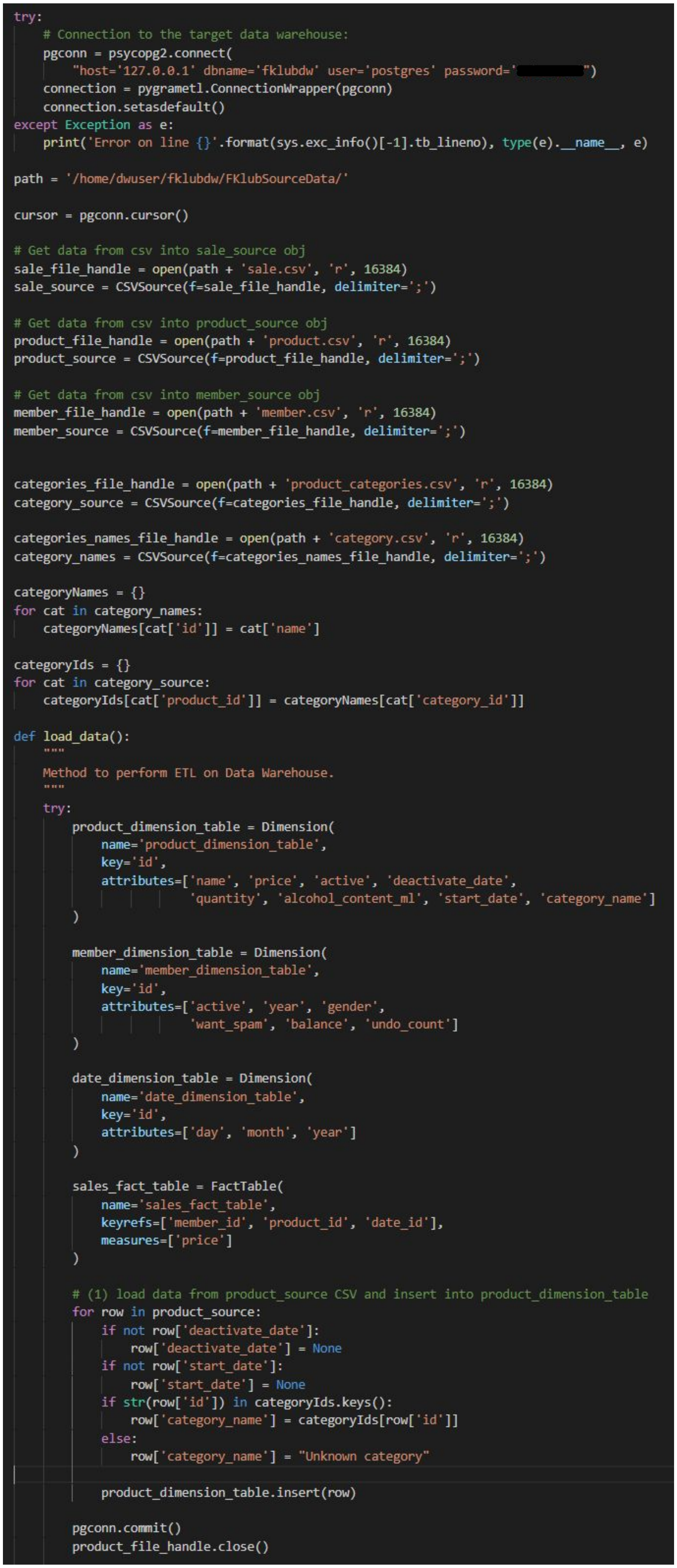
Python Scripts:

The following figures illustrate all Python scripts used for performing ETL initial load as well the creation of PostgreSQL tables.



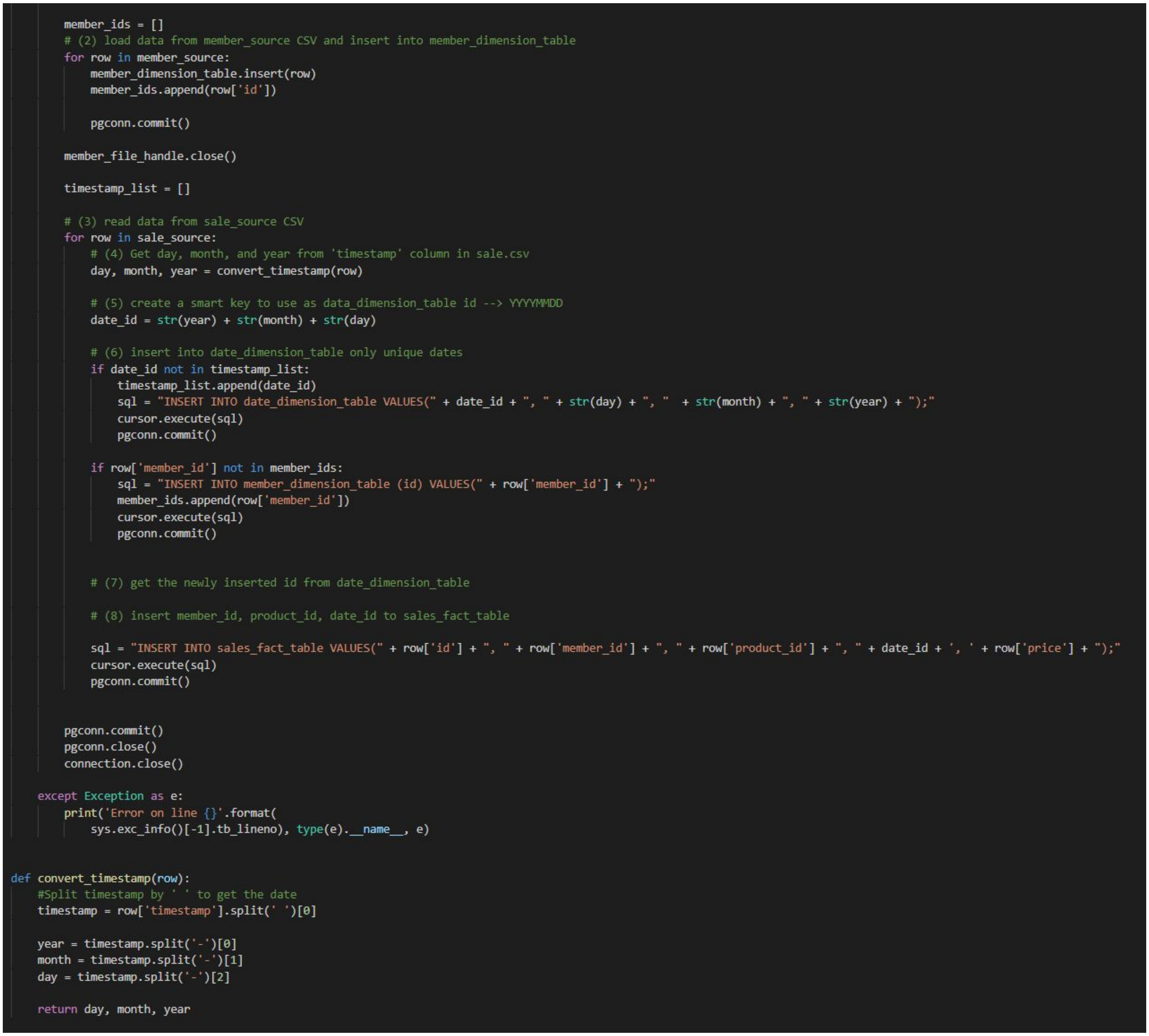
```
def create_tables():
    """ Method to create DW tables for Fklub. """
    commands = (
        """
        DROP TABLE IF EXISTS member_dimension_table cascade;
        CREATE TABLE IF NOT EXISTS member_dimension_table (
            id bigint NOT NULL PRIMARY KEY,
            active BOOLEAN,
            year VARCHAR(4),
            gender CHAR(1),
            want_spam BOOLEAN,
            balance BIGINT,
            undo_count BIGINT
        );
        """,
        """
        DROP TABLE IF EXISTS product_dimension_table cascade;
        CREATE TABLE IF NOT EXISTS product_dimension_table (
            id bigint NOT NULL PRIMARY KEY,
            name character varying(64) COLLATE pg_catalog."default" NOT NULL,
            price bigint NOT NULL,
            active boolean NOT NULL,
            deactivate_date timestamp with time zone,
            quantity bigint NOT NULL,
            alcohol_content_ml double precision,
            start_date date,
            category_name VARCHAR(64)
        );
        """,
        """
        DROP TABLE IF EXISTS date_dimension_table cascade;
        CREATE TABLE IF NOT EXISTS date_dimension_table (
            id bigint NOT NULL PRIMARY KEY,
            day bigint,
            month bigint,
            year bigint
        );
        """,
        """
        DROP TABLE IF EXISTS sales_fact_table cascade;
        CREATE TABLE IF NOT EXISTS sales_fact_table (
            id bigint NOT NULL PRIMARY KEY,
            member_id bigint,
            product_id bigint,
            date_id bigint,
            price bigint,
            FOREIGN KEY (member_id) REFERENCES member_dimension_table(id),
            FOREIGN KEY (date_id) REFERENCES date_dimension_table(id),
            FOREIGN KEY (product_id) REFERENCES product_dimension_table(id)
        );
        """
    )
    try:
        # connection to PostreSQL server
        pgconn = psycopg2.connect(
            "host='127.0.0.1' dbname='fklubdw' user='postgres' password='[redacted]'")
        connection = pygrametl.ConnectionWrapper(pgconn)
        connection.setasdefault()

        """
        cur = connection.cursor()
        for command in commands:
            cur.execute(command)
        # close communcation with the PostgreSQL database server
        cur.close()
        """
        cursor = pgconn.cursor()
        for command in commands:
            cursor.execute(command)
        pgconn.commit()

        # commit the changes
        connection.commit()
    except Exception as e:
        print('Error on line {}'.format(
            sys.exc_info()[-1].tb_lineno), type(e).__name__, e)
```

Figure 6 : Create PostgreSQL Tables)



```python
try:
    # Connection to the target data warehouse:
    pgconn = psycopg2.connect(
        "host='127.0.0.1' dbname='fklubdw' user='postgres' password='[redacted]")
    connection = pygrametl.ConnectionWrapper(pgconn)
    connection.setasdefault()
except Exception as e:
    print('Error on line {}'.format(sys.exc_info()[-1].tb_lineno), type(e).__name__, e)

path = '/home/dwuser/fklubdw/FKlubSourceData/'

cursor = pgconn.cursor()

# Get data from csv into sale_source obj
sale_file_handle = open(path + 'sale.csv', 'r', 16384)
sale_source = CSVSource(f=sale_file_handle, delimiter=';')

# Get data from csv into product_source obj
product_file_handle = open(path + 'product.csv', 'r', 16384)
product_source = CSVSource(f=product_file_handle, delimiter=';')

# Get data from csv into member_source obj
member_file_handle = open(path + 'member.csv', 'r', 16384)
member_source = CSVSource(f=member_file_handle, delimiter=';')


categories_file_handle = open(path + 'product_categories.csv', 'r', 16384)
category_source = CSVSource(f=categories_file_handle, delimiter=';')

categories_names_file_handle = open(path + 'category.csv', 'r', 16384)
category_names = CSVSource(f=categories_names_file_handle, delimiter=';')

categoryNames = {}
for cat in category_names:
    categoryNames[cat['id']] = cat['name']

categoryIds = {}
for cat in category_source:
    categoryIds[cat['product_id']] = categoryNames[cat['category_id']]

def load_data():
    """
    Method to perform ETL on Data Warehouse.
    """
    try:
        product_dimension_table = Dimension(
            name='product_dimension_table',
            key='id',
            attributes=['name', 'price', 'active', 'deactivate_date',
                        'quantity', 'alcohol_content_ml', 'start_date', 'category_name']
        )

        member_dimension_table = Dimension(
            name='member_dimension_table',
            key='id',
            attributes=['active', 'year', 'gender',
                        'want_spam', 'balance', 'undo_count']
        )

        date_dimension_table = Dimension(
            name='date_dimension_table',
            key='id',
            attributes=['day', 'month', 'year']
        )

        sales_fact_table = FactTable(
            name='sales_fact_table',
            keyrefs=['member_id', 'product_id', 'date_id'],
            measures=['price']
        )

        # (1) load data from product_source CSV and insert into product_dimension_table
        for row in product_source:
            if not row['deactivate_date']:
                row['deactivate_date'] = None
            if not row['start_date']:
                row['start_date'] = None
            if str(row['id']) in categoryIds.keys():
                row['category_name'] = categoryIds[row['id']]
            else:
                row['category_name'] = "Unknown category"

            product_dimension_table.insert(row)

        pgconn.commit()
        product_file_handle.close()
```

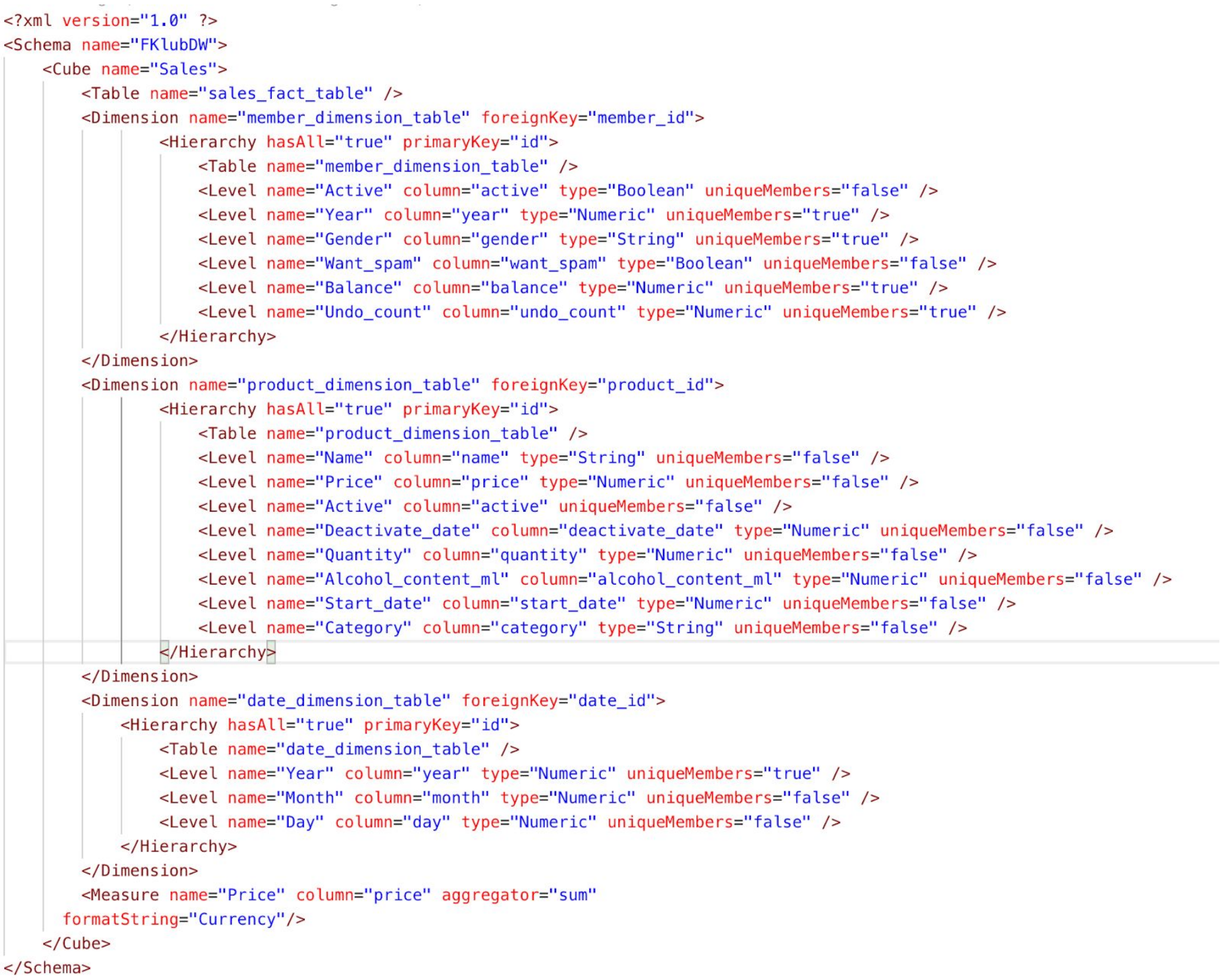
Figure 7 : load_data() method Part 1



```
        member_ids = []
        # (2) load data from member_source CSV and insert into member_dimension_table
        for row in member_source:
            member_dimension_table.insert(row)
            member_ids.append(row['id'])

            pgconn.commit()

        member_file_handle.close()

        timestamp_list = []

        # (3) read data from sale_source CSV
        for row in sale_source:
            # (4) Get day, month, and year from 'timestamp' column in sale.csv
            day, month, year = convert_timestamp(row)

            # (5) create a smart key to use as data_dimension_table id --> YYYYMMDD
            date_id = str(year) + str(month) + str(day)

            # (6) insert into date_dimension_table only unique dates
            if date_id not in timestamp_list:
                timestamp_list.append(date_id)
                sql = "INSERT INTO date_dimension_table VALUES(" + date_id + ", " + str(day) + ", "  + str(month) + ", " + str(year) + ");"
                cursor.execute(sql)
                pgconn.commit()

            if row['member_id'] not in member_ids:
                sql = "INSERT INTO member_dimension_table (id) VALUES(" + row['member_id'] + ");"
                member_ids.append(row['member_id'])
                cursor.execute(sql)
                pgconn.commit()


            # (7) get the newly inserted id from date_dimension_table

            # (8) insert member_id, product_id, date_id to sales_fact_table

            sql = "INSERT INTO sales_fact_table VALUES(" + row['id'] + ", " + row['member_id'] + ", " + row['product_id'] + ", " + date_id + ', ' + row['price'] + ");"
            cursor.execute(sql)
            pgconn.commit()


        pgconn.commit()
        pgconn.close()
        connection.close()

    except Exception as e:
        print('Error on line {}'.format(
            sys.exc_info()[-1].tb_lineno), type(e).__name__, e)


def convert_timestamp(row):
    #Split timestamp by ' ' to get the date
    timestamp = row['timestamp'].split(' ')[0]

    year = timestamp.split('-')[0]
    month = timestamp.split('-')[1]
    day = timestamp.split('-')[2]

    return day, month, year
```

Figure 8 : load_data() method Part 2

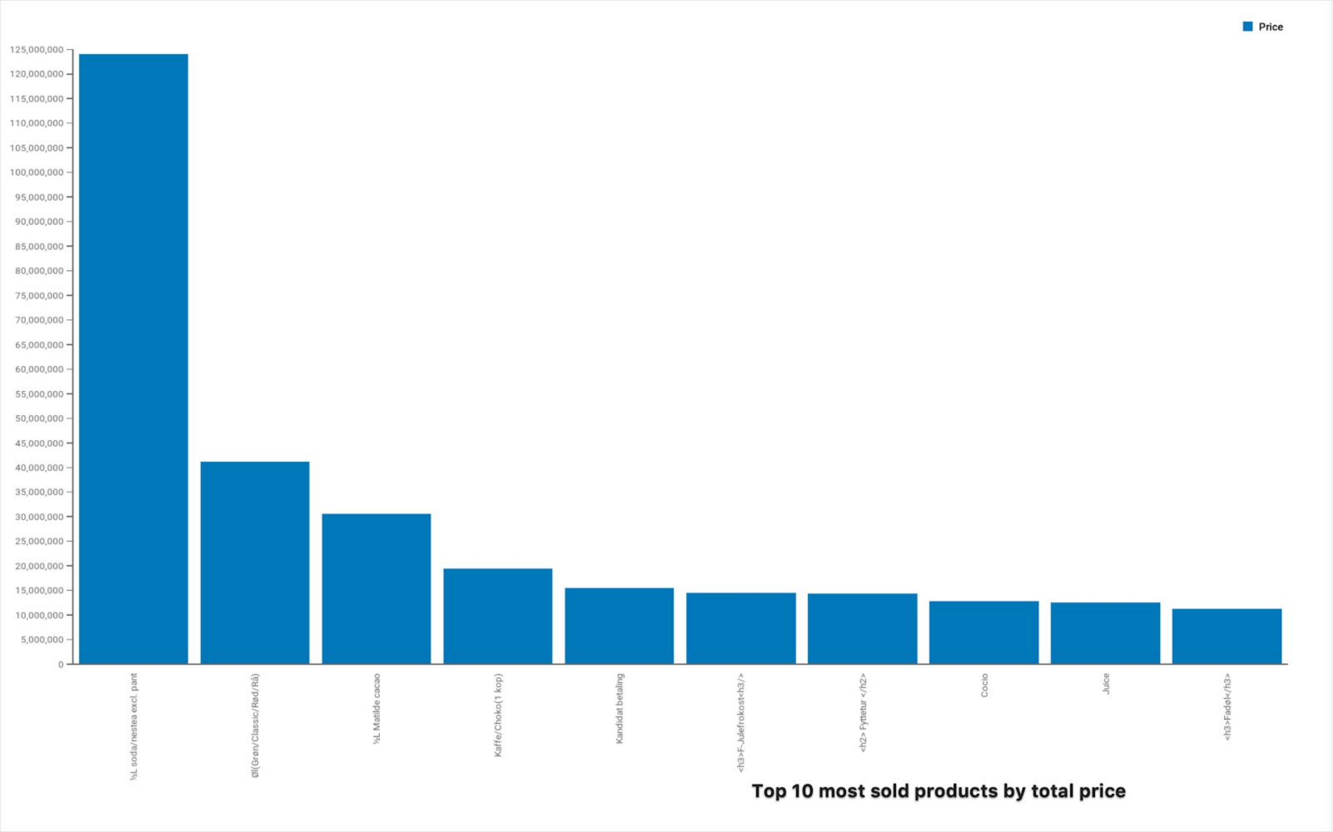
Task E



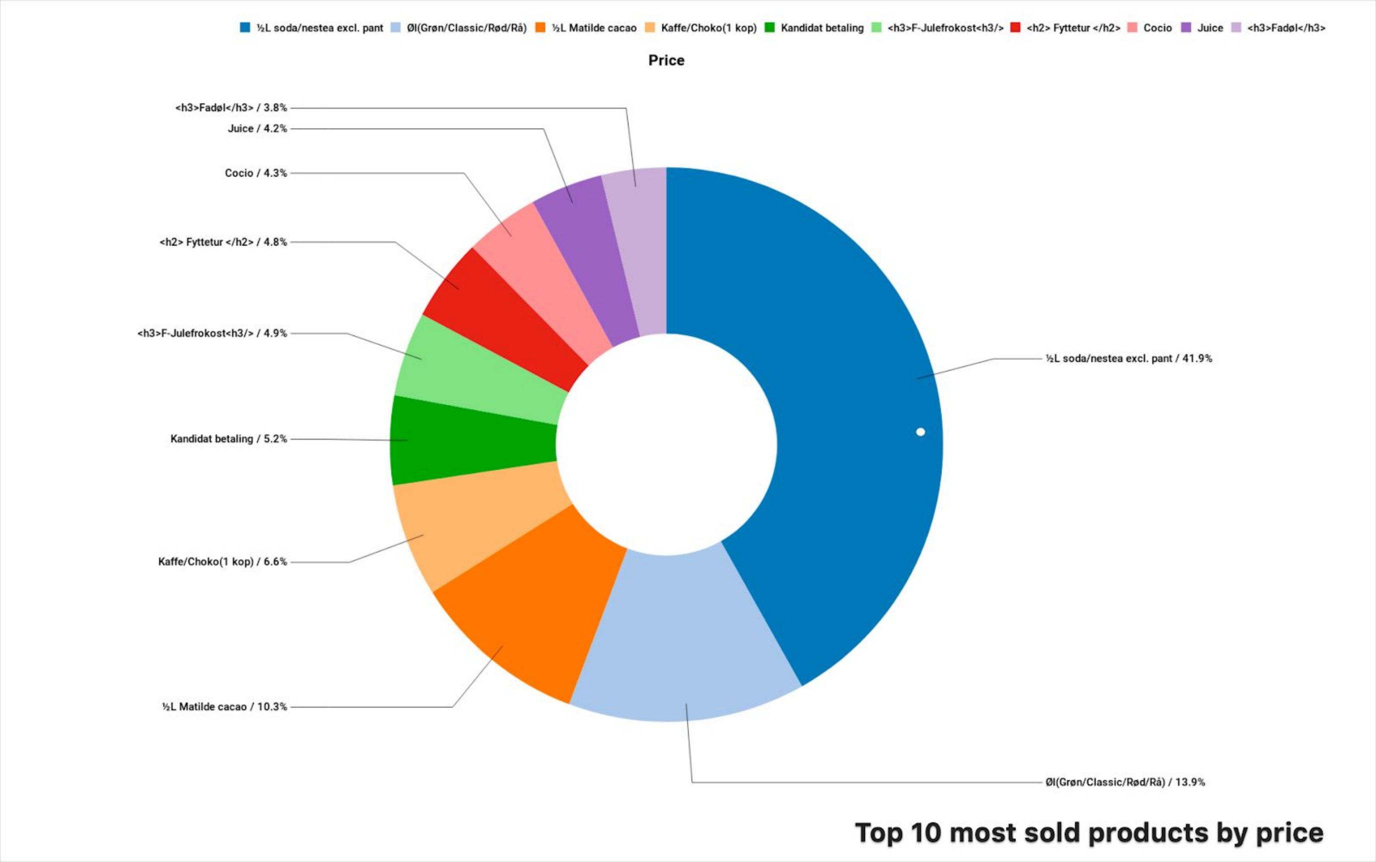
```
<?xml version="1.0" ?>
<Schema name="FKlubDW">
    <Cube name="Sales">
        <Table name="sales_fact_table" />
        <Dimension name="member_dimension_table" foreignKey="member_id">
                <Hierarchy hasAll="true" primaryKey="id">
                    <Table name="member_dimension_table" />
                    <Level name="Active" column="active" type="Boolean" uniqueMembers="false" />
                    <Level name="Year" column="year" type="Numeric" uniqueMembers="true" />
                    <Level name="Gender" column="gender" type="String" uniqueMembers="true" />
                    <Level name="Want_spam" column="want_spam" type="Boolean" uniqueMembers="false" />
                    <Level name="Balance" column="balance" type="Numeric" uniqueMembers="true" />
                    <Level name="Undo_count" column="undo_count" type="Numeric" uniqueMembers="true" />
                </Hierarchy>
        </Dimension>
        <Dimension name="product_dimension_table" foreignKey="product_id">
                <Hierarchy hasAll="true" primaryKey="id">
                    <Table name="product_dimension_table" />
                    <Level name="Name" column="name" type="String" uniqueMembers="false" />
                    <Level name="Price" column="price" type="Numeric" uniqueMembers="false" />
                    <Level name="Active" column="active" uniqueMembers="false" />
                    <Level name="Deactivate_date" column="deactivate_date" type="Numeric" uniqueMembers="false" />
                    <Level name="Quantity" column="quantity" type="Numeric" uniqueMembers="false" />
                    <Level name="Alcohol_content_ml" column="alcohol_content_ml" type="Numeric" uniqueMembers="false" />
                    <Level name="Start_date" column="start_date" type="Numeric" uniqueMembers="false" />
                    <Level name="Category" column="category" type="String" uniqueMembers="false" />
                </Hierarchy>
        </Dimension>
        <Dimension name="date_dimension_table" foreignKey="date_id">
            <Hierarchy hasAll="true" primaryKey="id">
                <Table name="date_dimension_table" />
                <Level name="Year" column="year" type="Numeric" uniqueMembers="true" />
                <Level name="Month" column="month" type="Numeric" uniqueMembers="false" />
                <Level name="Day" column="day" type="Numeric" uniqueMembers="false" />
            </Hierarchy>
        </Dimension>
        <Measure name="Price" column="price" aggregator="sum"
      formatString="Currency"/>
    </Cube>
</Schema>
```

Task F

The following graphs show the answers to the queries we considered in Task A.

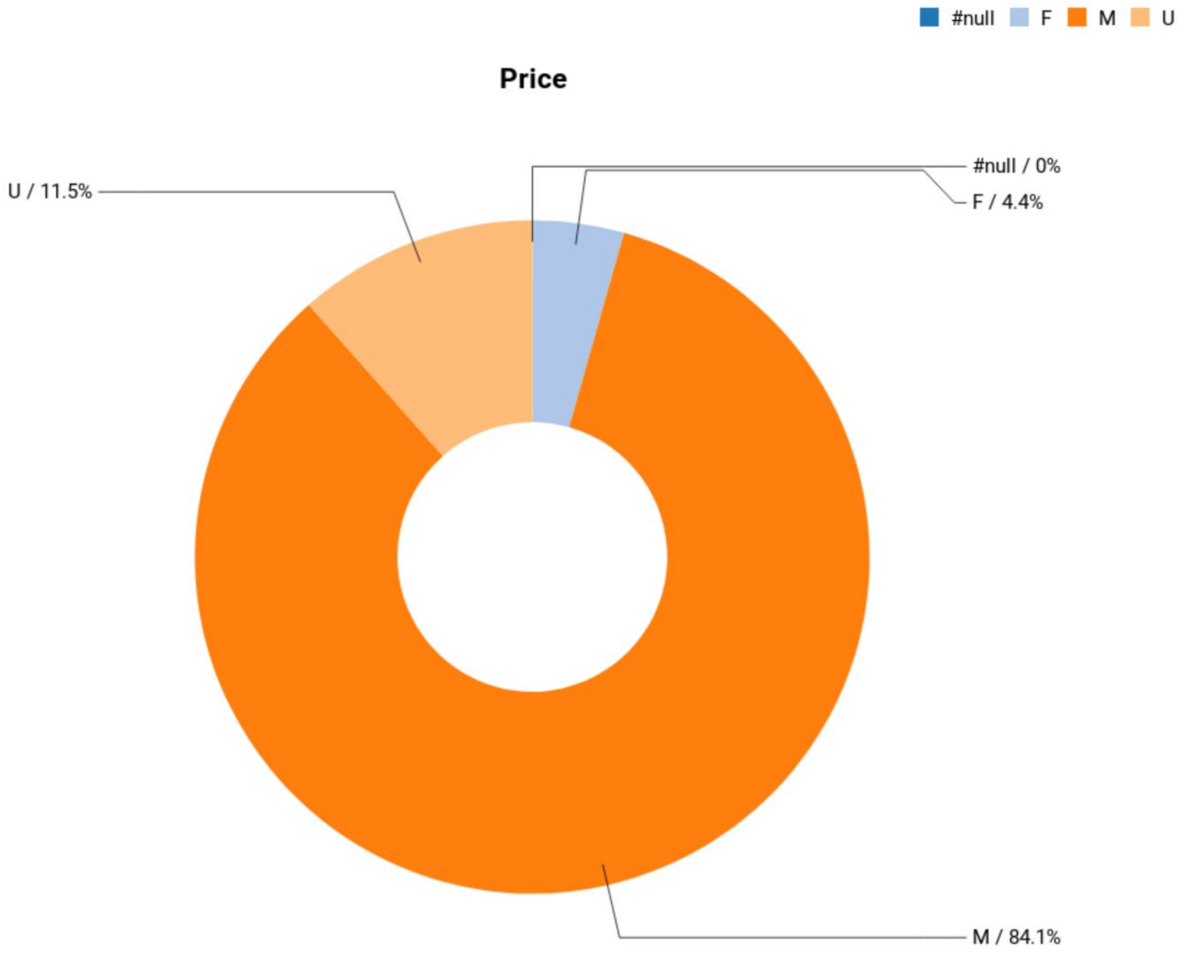


Top 10 most sold products by total price



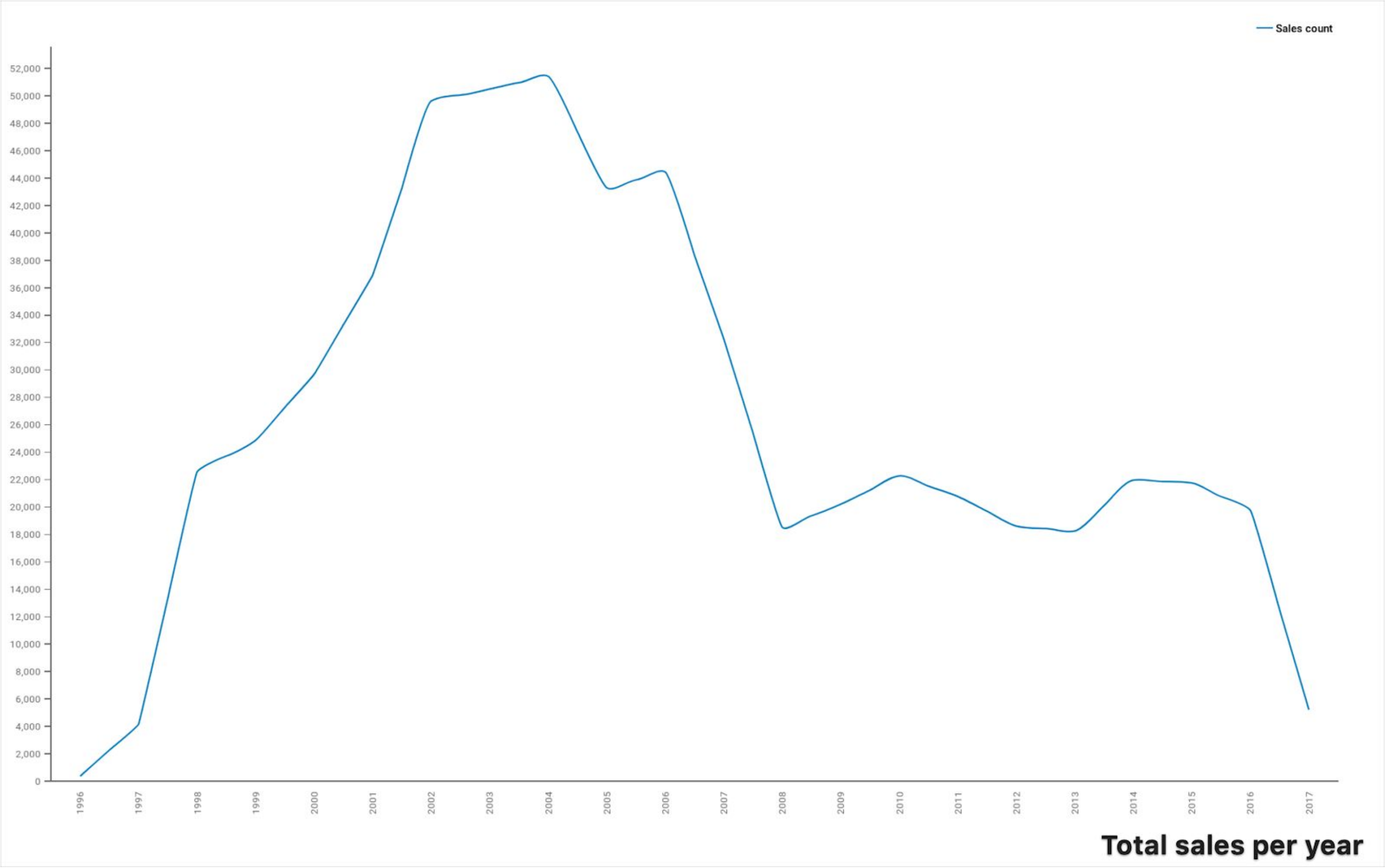
Top 10 most sold products by price

These two graphs show that almost half of the income of the FKlub comes from the sales of Soda.

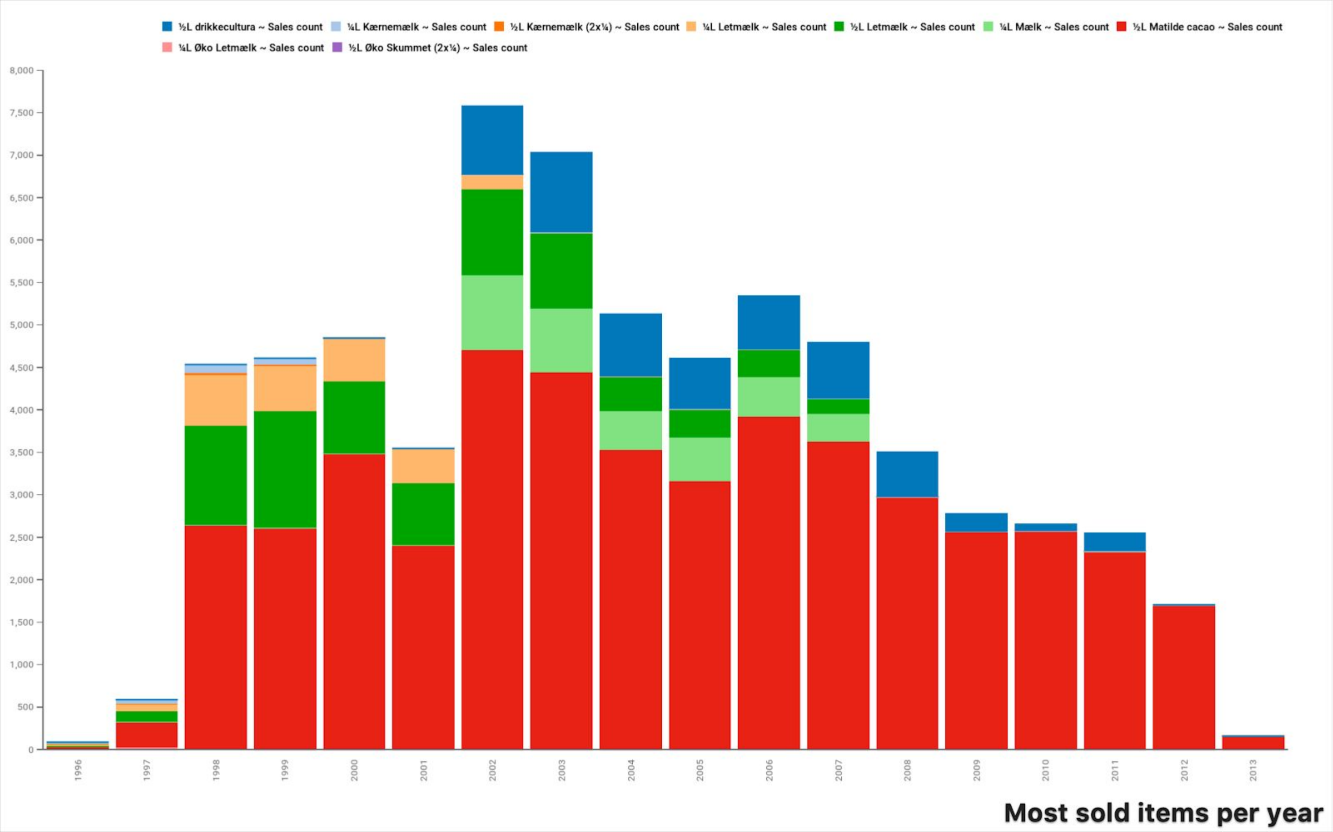


Money spent per member gender

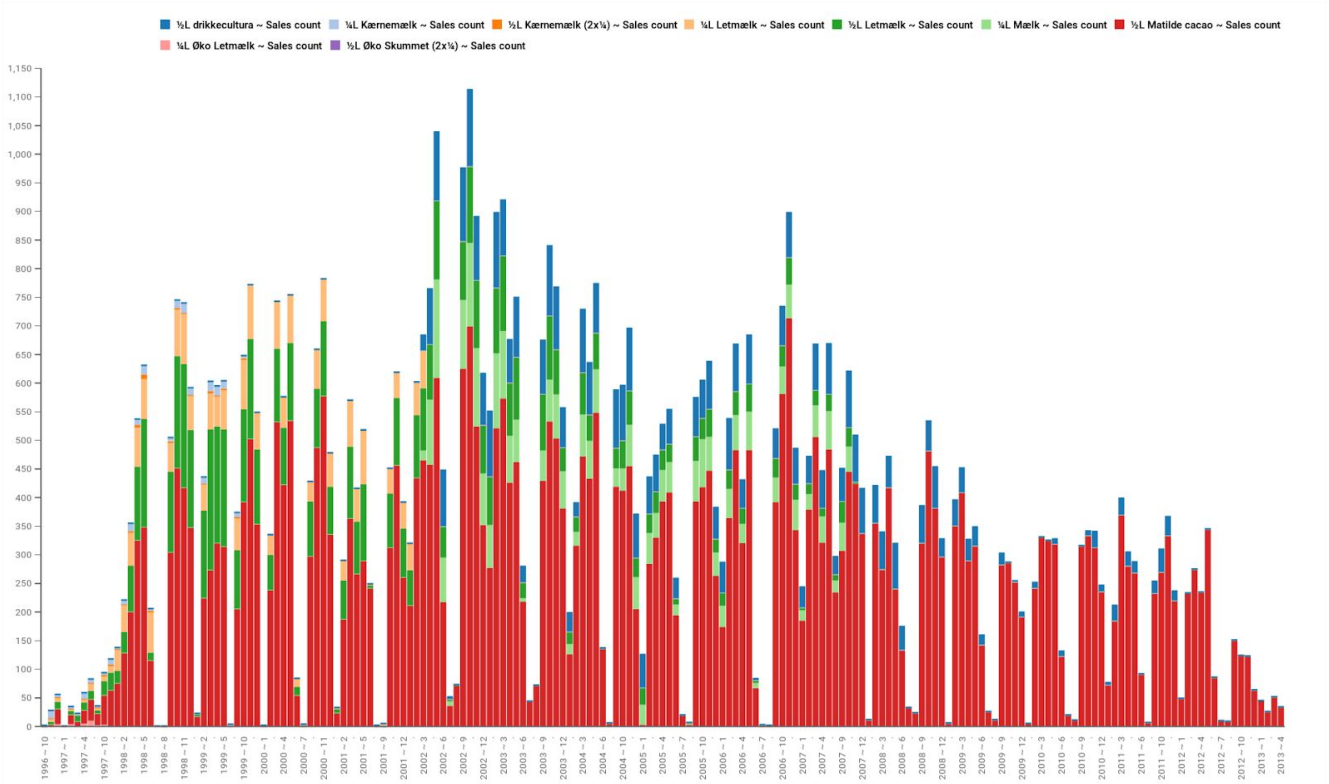
This pie chart shows the money spent per member gender on products sold by the FKlub. It highlights that most of the buyers are men. Moreover, despite the lack of data quality, the fact that a non negligible percentage of members are of undetermined sex doesn't affect our analysis because even if they were female, the men gender would always outnumber the female gender.



Since 1996, we can observe that the yearly income of the FKlub has sharply increased in the first few years, until reaching its peak in 2004 (52,000 DKK). As quickly as it had risen, it collapsed, until becoming steady around 20,000 DKK per year. The 2017 income is not relevant yet, as the last data obtained was from April, which is too early to have a significant observation.



This diagram shows that each year, the most sold item is by far the Matilde cacao. However, as the “Top 10 most sold product by price” pie chart highlights, sales of Matilde cacao represents only 10% of the income of the FKlub. Therefore, increasing the price of this particular product, would be greatly beneficial for FKlub.



Top 10 most sold items per month

Finally, this last diagram shows the monthly repartition of the sales, considering the quantity of products. We can observe that during each year, the graph is following the same cycle : most sales happen during September and November, or February and May, whilst almost no items are sold during July-August (holidays) or January (exam period).