Storing and Retrieving Data

PartC

Diogo Rasteiro R20170826
Gustavo Brito R20170760
Mariana Albernaz R20170785
Marta dos Santos R20170770
Miguel Mateus R20170752
Ricardo Peixoto R20170756

Contents

Executive Summary

PartC was founded in 2020 by six students from NOVA and aims to bring affordable technology products and services to Portugal and, in the future, to the entire world. Initially, PartC focused on being an online shop, shipping their products to the whole country. However, they have recently purchased a bankrupt shop, PartD, to enter the physical market, selling their goods there and providing tech-related services.

Due to its tech-savvy founders, the company has put great effort into building a robust and efficient Information System. especially when it comes to their Database, which stores not only information about its customers, employees and purchases but also uses SQL Views to provide critical information in readable formats and SQL Triggers to automatically implement business logic and maintain the Database's quality.

DB's model itself has been carefully refined to follow the general guidelines of model design, allowing it to store all the information most efficiently.

The Views are used to summarize and organize the information from the Invoices of the recorded purchases, such as totals and individual items. The triggers, on the other hand, assure that information such as an Item's stock and an Invoice's total, as well as log events that happen.

During the acquisition of the new business, PartC acquired an Excel file with the old company's customers. Since the businesses were similar, it was decided that the information about the customers would be inserted into the database and treated as if they were PartC's. Due to the difference in formats, it was necessary to find a tool to act as the bridge between PartD's format and the one used in our database. To solve this, Pentaho was used, which provided a simple way to transform the old data, so it fits together with the new one.

Methodology

The methodology of making this assignment started by a brainstorm about what we thought was necessary for a database about technology products. By doing this, we wanted to have no mind restrictions when it came with the guidelines given in the scope of a database, analysing what we thought was crucial to any online store databases.

After that, we gathered our ideas on the relationships between the entities to make the entity-relationship model. This stage also represented the beginning of the store PartC, as we reached some conclusions on the products, promotions, logistics and distribution channels that needed to be included in our schema.

We then proceeded to make the mandatory triggers, queries, and views which we will elaborate on a more advanced stage of this report. And since we had acquired a new company, Part D, that managed their clients with an excel sheet, we resorted to Pentaho to correctly perform a data integration process and include all client information in our database.

Following the required parts of this project, we decided to check if our database was correctly made. We conducted this analysis based on the 3 Normalization Forms lectured in class.

As part of extra project activities, we decided that we would make some extra chores for this project. As so, we decided to also include a trigger that would calculate and update the total price of the invoice based on the product prices and quantities, every time a new item is inserted on an invoice.

ERD Model

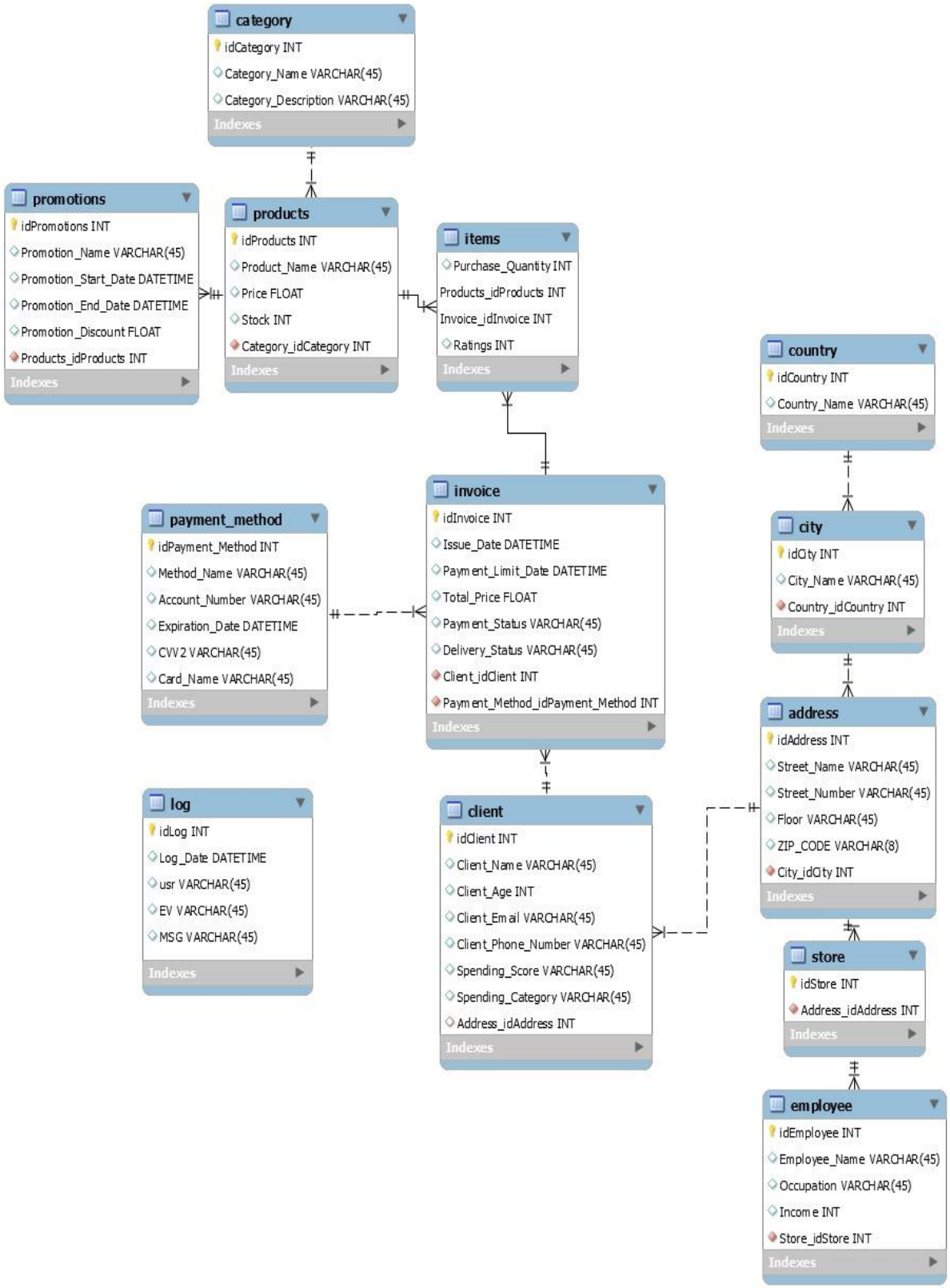

Figure 1 – ERD Model

Triggers

A trigger that updates the stock of products after the customer completes an order:

```
DROP trigger IF  EXISTS ITEMS_BI_UPDATE_STOCK;
DELIMITER $$
CREATE TRIGGER ITEMS_BI_UPDATE_STOCK
BEFORE INSERT
ON items
FOR EACH ROW
BEGIN

        UPDATE products as p, items i
        SET p.Stock = IF(p.stock>=new.Purchase_Quantity,p.Stock - NEW.Purchase_Quantity, p.stock)
        WHERE NEW.Products_idProducts = p.idProducts;

END $$
DELIMITER ;
```

Figure 2 – Trigger 1

A trigger that inserts a row in a “log” table if the price of a product is updated:

```
DROP trigger IF  EXISTS PRODUCTS_AU_INSERT_LOG;
-- Trigger 2
DELIMITER $$
CREATE TRIGGER PRODUCTS_AU_INSERT_LOG
AFTER UPDATE
ON products
FOR EACH ROW
BEGIN

    INSERT INTO log(Log_Date, usr, EV, MSG) values
    (now(), user(), concat(OLD.Price, ' - ', NEW.Price), concat('update product ', NEW.Product_Name));

END $$
DELIMITER ;
```

Figure 3 – Trigger 2

A trigger that calculates (updates) the price of an invoice (extra):

As previously stated, this trigger had the function of calculating and updating the Total price of an invoice, every time an item was added to it. Based on the product price, and the desired quantity, the item total (product price*quantity) was added to the previous value of the total price.

```
DROP trigger IF  EXISTS Invoice_Total_Price_Update;
-- Trigger 3
DELIMITER $$
CREATE TRIGGER Invoice_Total_Price_Update
After Insert
ON items
FOR EACH ROW
BEGIN
    UPDATE Invoice as inv, items i, products p
    SET inv.Total_Price= inv.Total_Price + (new.Purchase_Quantity*p.Price)
    WHERE NEW.Products_idProducts = p.idProducts and new.Invoice_idInvoice=inv.idInvoice;

END $$
DELIMITER ;
```

Figure 4 – Trigger 3

We decided to make another **extra trigger** that attributes a spending score filling the column Spending Category based on the Spending Score:

```
DROP TRIGGER IF EXISTS Client_Spending_Category_Update;
-- Trigger 4
DELIMITER $$
CREATE TRIGGER Client_Spending_Category_Update
BEFORE INSERT
ON partc.client
for each row
BEGIN

set NEW.Spending_Category=IF(NEW.Spending_Score<20,1, IF(NEW.Spending_Score<40,2,IF(NEW.Spending_Score<60,3,IF(NEW.Spending_Score<80,4,5))));

END $$
DELIMITER ;
```

Figure 5 – Trigger 4

Queries

1. List all the customer's names, dates, and products or services bought by these customers in a range of two dates.

We decided to use as a range of dates the 1st of January of 2020 until the current date.

```
SELECT Client_Name as 'Client Name', Issue_Date as 'Invoice Issue Date',
Product_Name as 'Product Name'
FROM client c, invoice i,  items it, products p
WHERE c.idClient =i.Client_idClient and
i.idInvoice = it.Invoice_idInvoice and
it.Products_idProducts = p.idProducts and
`Issue_Date` > '2020-01-01 00:00:00' and
`Issue_Date` < curdate();
```

Figure 6 – Query 1

2. List the best three customers (you are free to select the criteria that define a "best customer")

We decided that the best customer was the one who spent the most on our shop.

```
SELECT Client_Name as 'Client Name' , SUM(Total_Price) as 'Total Amount Spent'
FROM client c,  invoice i
WHERE c.idClient =i.Client_idClient
group by c.Client_Name
ORDER BY SUM(Total_Price)  DESC LIMIT 3;
```

Figure 7 – Query 2

3. Get the average amount of sales (in euros) by month and by year for the whole sales history. This is, to calculate the average by month you need to divide all the total sales by the total number of months.

Firstly, we decided to create this query by following the directions fully, getting the yearly and monthly average by dividing by all the years and months between the first and last sale:

```
SELECT CONCAT(EXTRACT(YEAR_MONTH FROM MIN(I.Issue_Date)),' - ',
EXTRACT(YEAR_MONTH FROM MAX(I.Issue_Date))) as 'Period of Sales',
SUM(I.Total_Price) AS 'TotalSales(euros)',
(SUM(I.Total_Price) / (YEAR(MAX(I.Issue_Date)) - YEAR(MIN(I.Issue_Date)))) as 'YearlyAverage',
(SUM(I.Total_Price) / (((YEAR(MAX(I.Issue_Date)) - YEAR(MIN(I.Issue_Date)))) * 12 +
MONTH(MAX(I.Issue_Date)) - MONTH(MIN(I.Issue_Date)))) as 'MonthlyAverage'
FROM Invoice AS I;
```

Figure 8 – Query 3

We also decided to do this query taking only into account the years and months where there were sales:

```
SELECT CONCAT(EXTRACT(YEAR_MONTH FROM MIN(I.Issue_Date)),' - ',
EXTRACT(YEAR_MONTH FROM MAX(I.Issue_Date))) as 'Period of Sales',
SUM(I.Total_Price) AS 'TotalSales(euros)',
(SUM(I.Total_Price) / COUNT(DISTINCT YEAR(I.Issue_Date))) as 'YearlyAverage',
(SUM(I.Total_Price) / COUNT(DISTINCT EXTRACT(YEAR_MONTH FROM I.Issue_Date))) as 'MonthlyAverage'
FROM Invoice AS I;
```

Figure 9 – Trigger 3.1

4. Get the total sales by geographical location (city/country).

By city:

```
SELECT c.City_Name as 'City',
COUNT(i.idInvoice) as 'Total Number of Sales',
SUM(i.Total_Price) as 'Total Value of Sales'
From City c
INNER JOIN  Address as a ON c.idCity = a.City_idCity
INNER JOIN `Client` as cl ON a.idAddress = cl.Address_idAddress
INNER JOIN Invoice as i ON cl.idClient = i.Client_idClient
GROUP BY c.City_Name;
```

Figure 10 – Trigger 4

By country:

```
SELECT co.Country_Name as 'Country',
COUNT(i.idInvoice) as 'Total Number of Sales',
SUM(i.Total_Price) as 'Total Value of Sales'
From Country co
INNER JOIN City as c ON co.idCountry = c.Country_idCountry
INNER JOIN  Address as a ON c.idCity = a.City_idCity
INNER JOIN `Client` as cl ON a.idAddress = cl.Address_idAddress
INNER JOIN Invoice as i ON cl.idClient = i.Client_idClient
GROUP BY co.Country_Name;
```

Figure 11 – Trigger 4.1

5. List all the locations where products/services were sold and the product has customer's ratings.

```
SELECT c.City_Name as 'City'
From City c
INNER JOIN  Address as a ON c.idCity = a.City_idCity
INNER JOIN `Client` as cl ON a.idAddress = cl.Address_idAddress
INNER JOIN Invoice as i ON cl.idClient = i.Client_idClient
INNER JOIN Items as it ON i.idInvoice = it.Invoice_idInvoice
WHERE it.Ratings IS NOT NULL
GROUP BY c.City_Name;
```

Figure 12 – Trigger 5

Views

Totals

As for the views we made two of them, each related to a different part of the invoice.

The first one demonstrated the Clients information, meaning the invoice id (number), the clients name his street address, city, state and country and his zip code.

```
create view invoice_head
as select  i.idInvoice,i.Total_Price,i.Issue_Date, c.Client_Name, a.Street_Name,  ci.City_Name, cou.Country_Name
from invoice i
join client c on i.Client_idClient=c.idClient
join address a on c.Address_idAddress=a.idAddress
join city ci  on a.City_idCity=ci.idCity
join country  cou on ci.Country_idCountry=cou.idCountry
group by i.idInvoice
;
```

Figure 13 – View 1

Details

The second one checks the detailed information on each invoice. Namely, the quantity and price of each item bought in a particular invoice. Due to the nature of SQL, this prints each item bought with its information and sorts them by the Invoice's ID.

```
-- 2. View 2
DROP VIEW IF EXISTS invoiceDetails;
CREATE VIEW invoiceDetails AS
    SELECT
        items.Invoice_idInvoice AS 'Invoice',
        products.Product_Name AS 'Description',
        CONCAT(products.Price, '€') as 'Unit Cost',
        items.Purchase_Quantity as 'Quantity',
        CONCAT(items.Purchase_Quantity * products.Price, '€') as 'Total Cost of Item'

    FROM
        partc.invoice,
        partc.items,
        partc.products
    WHERE
        invoice.idInvoice = items.Invoice_idInvoice
            AND products.idProducts = items.Products_idProducts
    ORDER BY idInvoice ASC
```

Figure 14 – View 2

Acquisition of PartD

Part D was a competitor company that PartC, our company, decided to acquire. Surprisingly, despite the fact they were an informatics company, they still had their database as an excel sheet, due to its small company size.

To transfer their client data to our database, we resorted to Pentaho for a safe and integral process of Data Integration.

It is important to refer that the client table stored in the Excel Sheet had only the self-explanatory fields, customer name, age and spending score.

To fit our database requirements multiple transformations were required.

The first step was to load the Excel data into Pentaho using the function of Microsoft Excel input. Providing the file extension and its path we were able to load all columns and their info.

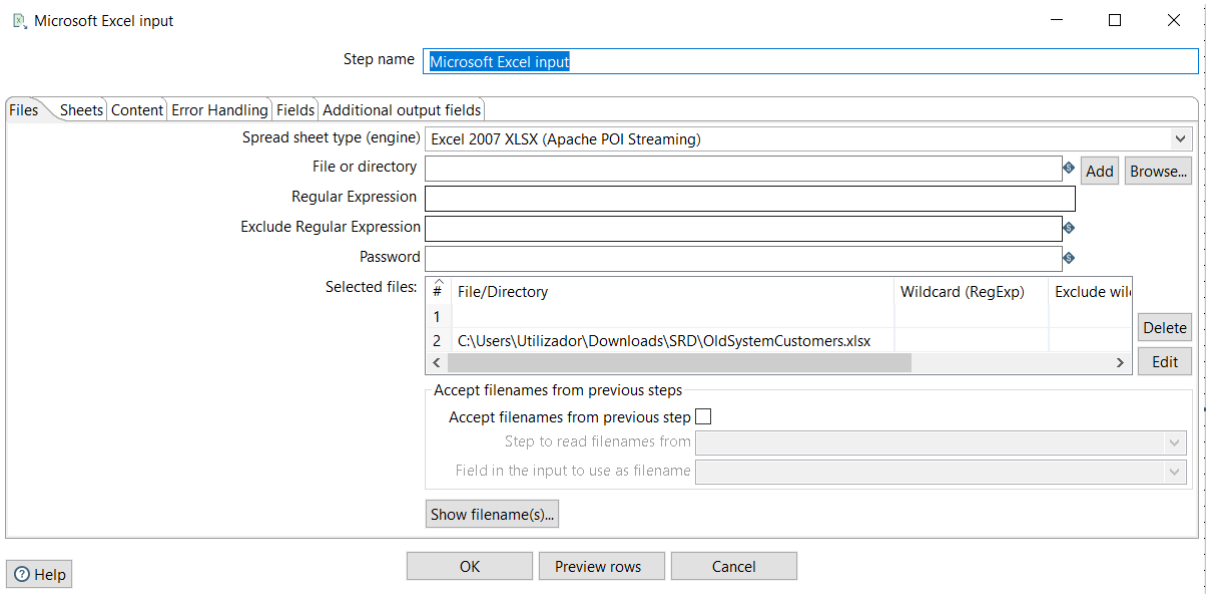

Figure 15 – Pentaho Input

After this simple step, we started our data processing. No one-letter names were allowed to identify a customer. For that reason, all clients with this problem were not kept in our database. To achieve this, we used the function Calculator, the calculate the number of letters, length, of each customer name. The result was kept on one column called ‘length’.

Marketing Campaign

We decided to create three marketing campaigns that will allow the integration of Part D to function more smoothly. We will see them in more detail below.

Figure 16 – Marketing Campaign 1 (PartD Customers)

This marketing campaign is directed to PartD's customers and we will e-mail PartD's old clients (information from the Pentaho methodology) a coupon to make sure that we make ourselves visible and that the clients that perhaps were prone to buy from PartD know that they can count on us as well.

The second marketing campaign will also be made by e-mail and is more directed in a customer-value perspective and will be applied to our old customers.

Figure 17 – Low Value Customers Campaign

This e-mail will be sent to the customers that are ranked “1” and “2” on the “Spending_Category” attribute in our database. With a single select, we will find the emails of the customers that fall in these categories. This promotion wants to make “weak” customers buy our products, and as so we are giving them reasons to come back.

On the reverse perspective, the customers that are ranked “5” in “Spending_Category” will receive a proper e-mail with the following message:

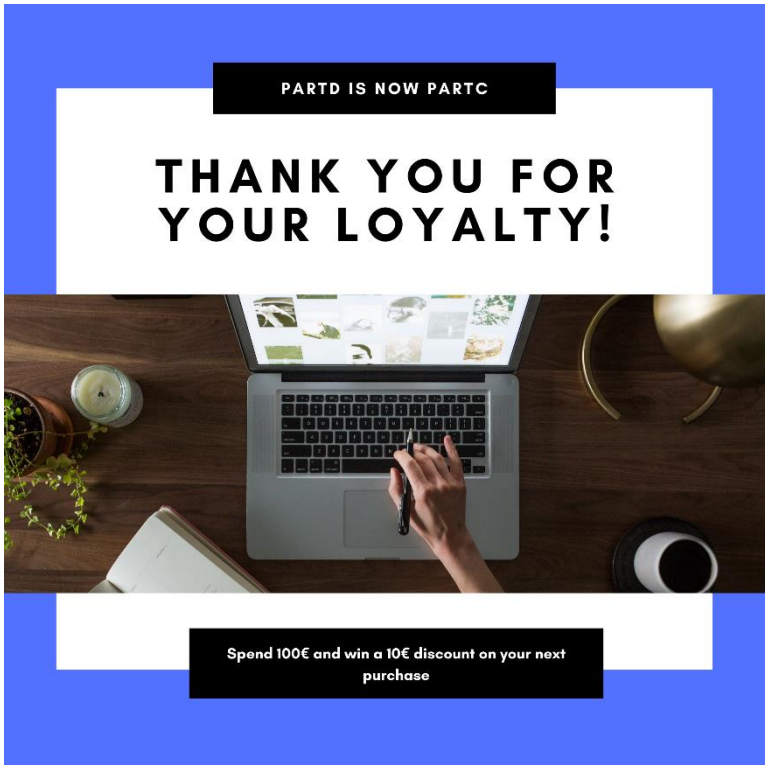

Figure 18 – High Value Customers Campaign

The trust of this customers seems to be already gained in our brand, and as so we will reward them with some points in case they do purchase some more products!