Mario’s Pizza Data Overhaul

Design, data consolidation and analysis

Fontys Hogescholen
Semester 4 – Data Processing – Module 1
Nov 2021

Minka Firth
Rob Lavrijssen
Colin Pieper

Contents

Case

The pizza restaurant chain Mario’s Pizzas is rapidly expanding. This expansion has resulted in various organizational and technical challenges. One of the serious difficulties inhibiting further growth is related to the data organization and storage of Mario’s. To overcome this obstacle, Mario’s is trying to professionalize their data storage and structures. As an inspiration, the company is mainly looking at one of their major competitors, Domino’s Pizzas. Mario’s wants to analyze and imitate its competitor’s data structure, design and implement this new structure, consolidate its data into this new structure and analyze its own data to answer various business questions.

Analysis and design

The first step was analyzing the website and order process of Domino’s (see appendix for various raw notes that were taken during this process). Based on this analysis and the existing data from Mario’s, we designed a new database structure that would be suitable for Mario’s. The final design and the various iterations are attached in the ‘ERD’-folder included with this report.

Implementation of design

The next step was converting this design into an actual implementation of a functioning database. First, some choices had to be made regarding the database management system.

Relational or non-relational

The first choice was to use a relational or non-relational database. We decided to go for a relational database. This makes it easier to understand the relationships among the available data and enables us to retrieve data in one or more tables with a single query. The main reason we chose to go for a relational database instead of a non-relational database is the ability to create meaningful information by joining the tables. Since these tables are interlinked, it makes updating these records a lot easier, thus maintaining uniform data in all of them. Data in relational databases are also normalized. This process enables the creation of constraints preventing the existence of duplicate records.

Database management system

Having decided on a relational database management system, the next question was deciding on a specific RDBMS. Various factors were considered and compared, such as performance, available tooling, cost and popularity.

The initial selection was done based on popularity of database engines. The top three, based on the DB-Engines ranking (which is based on various factors such as mentions and job offers), were Oracle, MySQL and Microsoft SQL Server.

MySQL and Microsoft SQL Server both offer free and paid options. MySQL Community Edition is free and open-source, but offers various tiers of paid editions with additional features and support. SQL Server also has a free tier, which is rather limited in capabilities, and has various paid tiers. The costs of the paid tiers are comparable, with SQL Server being somewhat more expensive. Oracle has a free edition, but its limitations are severe. The paid tiers are significantly more expensive than both SQL Server and MySQL. Due to this cost consideration, we decided to drop Oracle from consideration.

To evaluate performance, various benchmark reports and comparisons published online were compared. SQL Server seems to have a clear edge in this regard, especially with larger data sets, which is an important consideration due to Mario’s ambitions to expand over the coming years.

There’s excellent tooling available for both MySQL and SQL Server. There’s wide support for common programming languages and DBA tools. Due to our decision to use Entity Framework Core, motivated later in this report, SQL Server has a clear advantage.

Based on these factors, we decided to use the Microsoft SQL Server as Mario’s database management system.

Design implementation

Having decided on a database management system, we could now actually implement our design. We used SQL to define the various tables and columns of this design. The resulting SQL file can be found in the Database-folder attached with this report.

Data consolidation and import

Next up was consolidating all existing Mario’s Pizza’s data and importing this into the new data structure. The existing data was delivered in various file formats contained various errors. Mario’s will slowly start adopting the new data structure and systems, so the import method needs to be able to run automatically and update or skip data where necessary.

Approach

We considered various approaches to import all existing data. Further important considerations were that the data would have to be validated, rejected when failing validation, and logged for manual intervention. Based on this, we decided on building an import application using a modern programming language.

We decided to write our application in C# using the .NET 5 framework. It is a mature framework, so there will be a lot of documentation available online, and it has a lot of fitting features for this project.

Another major decision was whether to use an ORM, or use more integrated database features such as stored procedures and triggers. Potential benefits of the latter are potential increased performance. However, the performance benefit of using stored procedures has largely become insignificant for modern databases due to the increasing efficiency and optimization of the query planner and caching (when using parameterized queries). A major downside of using stored procedures and triggers is that you split your business logic in several places, increasing maintenance burden.

Based on these factors, we decided against using stored procedures and triggers. Instead, we opted for an approach using an ORM. .NET's ORM (Entity Framework Core) was created by Microsoft, just like MSSQL, which provides advanced and optimized integration. This allows us to write our logic in C# instead of raw SQL, allowing us to use the full range of features modern frameworks provide, decreasing development time and increasing ease of maintenance.

Additionally, a part of the import was created in a separate Java using raw SQL queries, to split the workload more easily between the group members and provide a more familiar development environment.

Result

The result were two applications, which when combined are able to validate, transform and import Mario’s data into the new database.

.NET import application

The import application written in .NET is a console application which can process Mario's pizzas, pizza sizes, ingredients, miscellaneous products, postal codes and orders. The application can run both interactively and non-interactively, and can be automated using for example a cronjob. The supplied data goes through several stages:

1. Loading/parsing, where the supplied data file is loaded into the application and converted into models reflecting the structure the data is supplied in
2. Cleaning, where any data can be corrected or cleaned, for example the removal of currency symbols and whitespaces
3. Validation, where data is checked for validity and rejected when not conforming to the business rules
4. Transformation, where data is transformed into models reflecting Mario's new data structures
5. Saving, where the transformed models are persisted in the database

Data failing validation will be skipped and be logged with a unique identifier for this data, such as a line number or key, and the field failing validation.

Saving inserts any new records and updates any existing records based on a unique key in the supplied data, for example the name of an ingredient. Orders are the exception; existing orders are skipped instead of updated, due to the immutability an order should have after being placed. Regardless of the type of data, no records are ever deleted during an import.

The source code is supplied in the 'Source/NET/MarioImport'-folder included with this report.

Java import application

The import application written in Java processes Mario's branches. The application goes through several stages:

1. Loading/parsing, where the supplied data file is loaded into the application and parsed (and cleaned) line-for-line. The several blocks of text representing one branch are identified, and the block is processed once all data representing one branch is complete
2. Processing, where the various parsed lines are identified based on the type of data it represents, cleaned, and validated
3. Saving, where the parsed data is either inserted into the database for new branches, or updated for existing branches

The source code is supplied in the 'Source/Java/MarioImport'-folder included with this report.

Final import application

Finally, to create one combined experience, we wrote a PowerShell-script. This script runs all imports in the correct order and logs the output of these imports into a combined log file, which is persisted to disk and can be checked at a later point. Currently, this import takes around two minutes in total. This duration decreases to around 90 seconds if the script is run a second time with the same data.

The source code is supplied in the 'Script/Import.ps1'-file.

Data analysis

Now that Mario's has consolidated all its import data into a new data structure, the company has new opportunities to analyze its data. This allows Mario's to make business decisions based on their actual data.

Visualization software

To make it easier to analyze the raw data and share these findings with the company, we looked at several options to visualize the data. There are several popular applications that exist for this purpose. Three well-known software suites for this purpose are Power BI, Looker and Tableau. To decide which software we were going to use, we made a list of pros and cons for each:

Power BI

Pros	**Cons**
Can use almost any form of data	Free version is limited, such as sharing dashboards and reports
There is a free version	
Has both a desktop and a cloud version	
Easy to use	
Professional-looking UI	

Tableau

Pros	**Cons**
Easy to use	Official version is very expensive
There is a free version	Security concerns
Multiple connections available	Very resource intensive

Looker

Pros	**Cons**
Simple UI	Has its own language, LookML
Unique form of visualization	Takes a lot of effort to set up
	Needs a lot of maintenance
	No free version available; only a trial

Conclusion

Based on these pros and cons, we decided to go with Power BI.

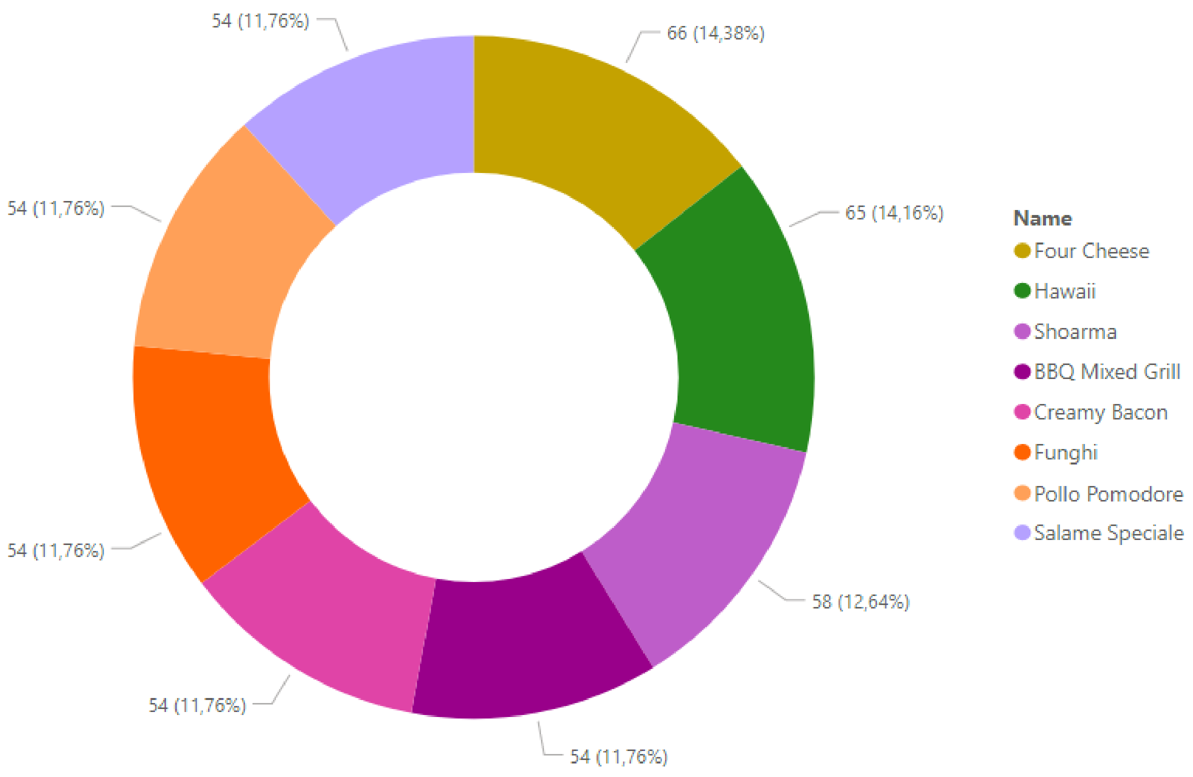
Business questions

Using various custom-made Power BI dashboards, we can easily analyze and combine data and use this to answer various business questions. As an example, we came up with five business questions that might be valuable to Mario's:

1. Which products are most popular in a certain branch?
2. During what time of year does Mario's generate the most revenue?
3. During this time of year, what products are the most frequently sold?
4. Which products are sold most often during Christmas?
5. Which branches generate the least amount of revenue?

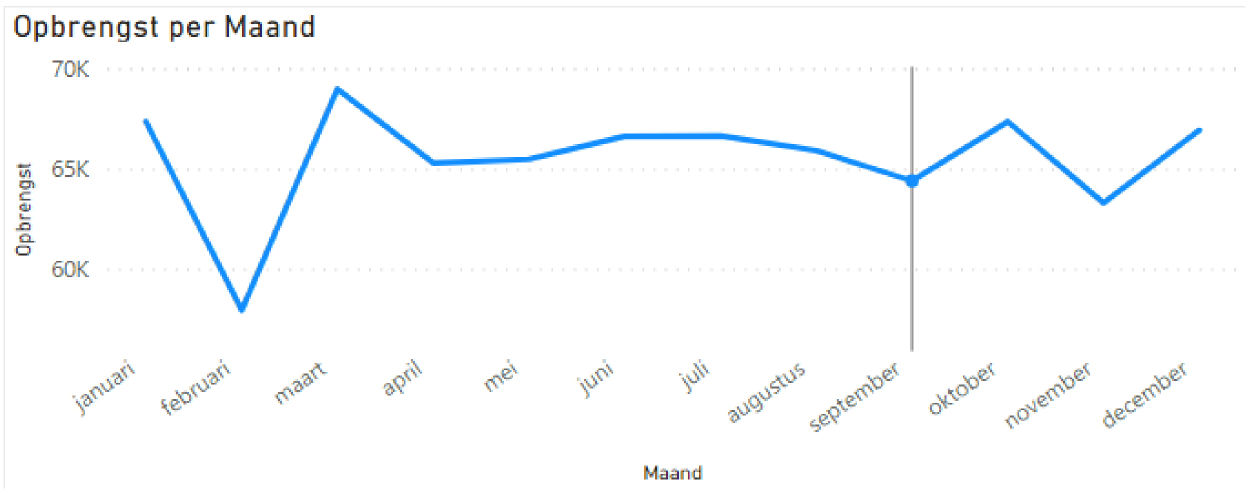
1. Which products are most popular in a certain branch?

For this example, we selected a random branch: Rotterdam Noord. The data shows that the most popular product sold is the Pizza Four Cheese, followed by the Pizza Hawaii and the Pizza Shoarma.



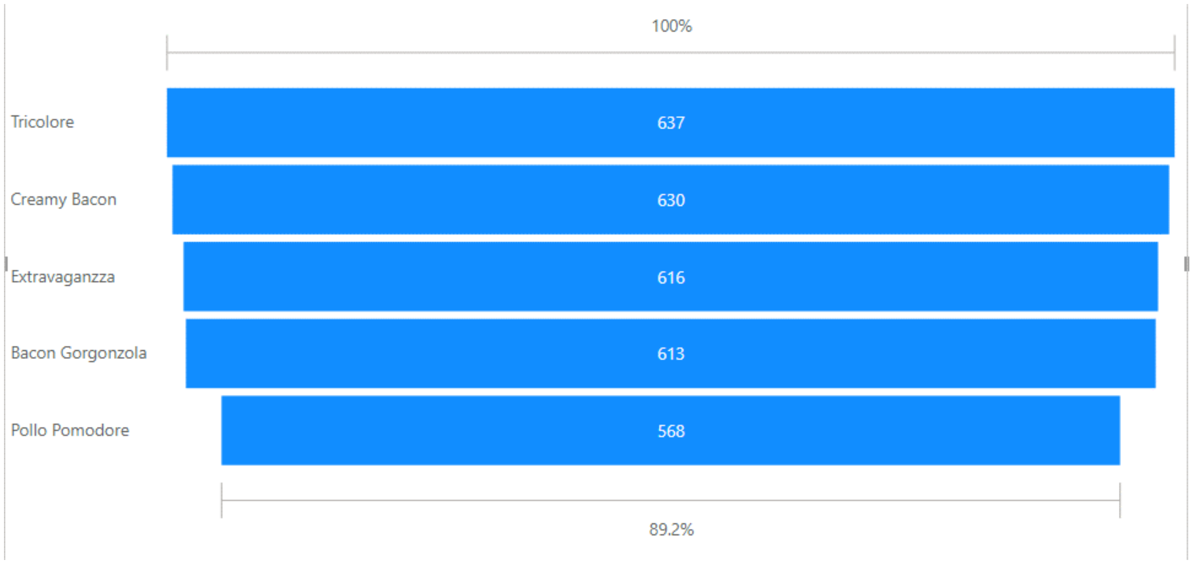
2. During what time of year does Mario's generate the most revenue?

Most revenue is generated in March, followed by January and October.



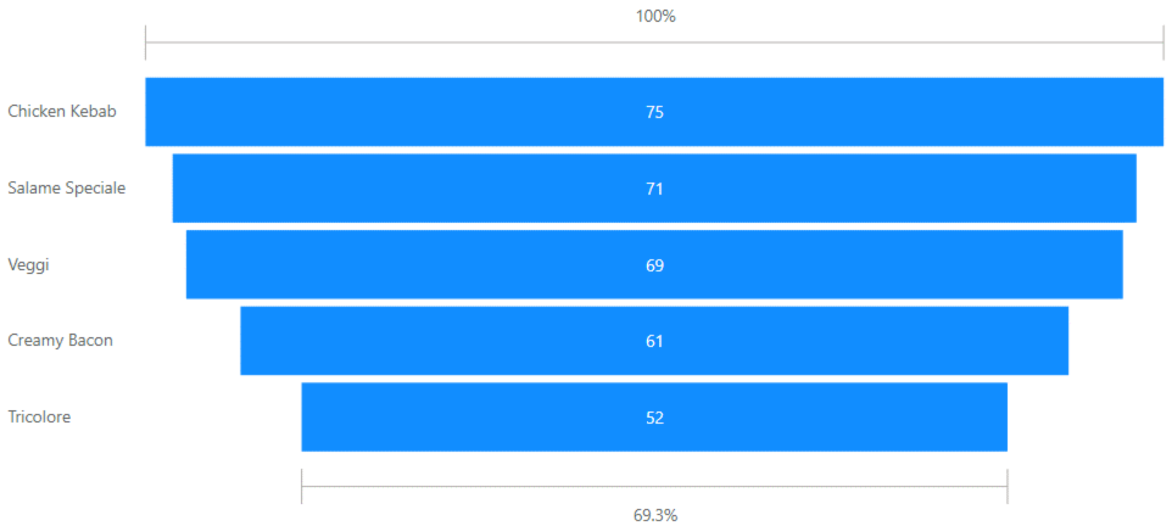
3. During this time of year, what products are the most frequently sold?

During March, the products sold most are the Pizza Tricolore, followed by Pizza Creamy Bacon.



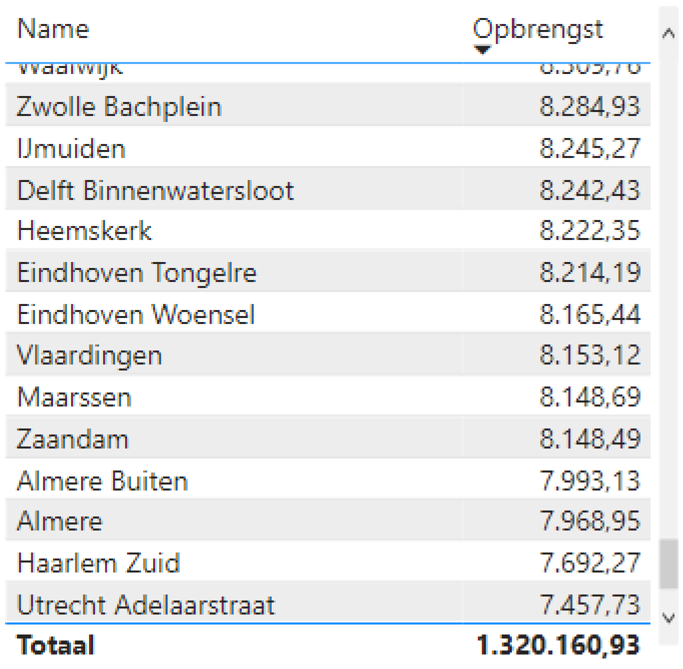
4. Which products are sold most often during Christmas?

During Dec 24-26, the products sold most often are the Pizza Chicken Kebab, Pizza Salame Speciale and Pizza Veggi.



5. Which branches generate the least amount of revenue?

The branches with the lowest revenue total during the measured period are Utrecht Adelaarstraat, Haarlem Zuid and Almere.



Name	Opbrengst
Waalwijk	8.309,76
Zwolle Bachplein	8.284,93
IJmuiden	8.245,27
Delft Binnenwatersloot	8.242,43
Heemskerk	8.222,35
Eindhoven Tongelre	8.214,19
Eindhoven Woensel	8.165,44
Vlaardingen	8.153,12
Maarssen	8.148,69
Zaandam	8.148,49
Almere Buiten	7.993,13
Almere	7.968,95
Haarlem Zuid	7.692,27
Utrecht Adelaarstraat	7.457,73
Totaal	**1.320.160,93**

Appendix: Raw notes from initial Domino's order process analysis

Notes #1

Week 1 - Analysis and database design

Analyse the Domino's Pizza website and elaborate/process this in a report/document. Take a good look at the functionalities of the site (order process etc.) and the diverse (dynamic) data. Make screenshots to support your analysis and findings. You've to do research on the information sources which are delivered from Mario. Finally all this data has to fit in a newly designed data structure like Domino's.

As a result of the analysis produce a database design which fits the structure of Domino's and the data sources of Mario.

Steekwoorden eerste analyse

- Artikelen
    - Pizza's
        - Ingrediënten (minimaal 1, maximaal 11, meerdere keer per ingrediënt, heeft meerprijs)
        - Grootte (bepaalt beschikbare bodemtypes)
        - Bodemtype
        - Saus
        - Half-half?!
    - Overige artikelen zonder keuzes
    - Overige artikelen met keuzes (bijv. saus)
- Bestelproces
    - Bezorgadres (straat, huisnr, postcode, plaats)
    - Vestiging
    - Naam
    - Mobiel nummer
    - E-mail
    - Bezorginstructies
    - Nieuwsbrief/SMS-berichten
    - Klantaccount
    - Promocode
    - Betaalinformatie
    - Bezorgtijd

Beperkingen/opmerkingen eerste ERD

- Meertaligheid? > losse table met talen en losse tables met vertalingen
- Half-half pizza?
- Beperkingen welke grootte welke bodemtype mag hebben
- Bezorgtijden

Notes #2

Analysis of Dominos

Different types of foods/dishes:

- Crunchy Chicken
- Side dishes
- Deserts
- Drinks
- Breads

  And their main dish: Pizza (or sandwiches) with various customization option available:

  - Sizes
  - Sauce
  - Crust
  - Ingredients
- Orders

  Consists of different aspects such as:
  - Branch where ordered
  - The Customer + phone number & email
  - Take away or delivery, +if Delivery Address
  - Whatever that has been ordered + product info
  - Payment method and price
  - Any notes for the deliverer
  - Delivery time

- Customers

We ran into a problem when we tried to just imagine a Customer with one Address, but Since A customer should be able to order deliveries to different addresses, we tried to look at it this way:

Analysis of given Data

Things that stood out

- No File for Customers or addresses -> All just part of the order Files.
- Each Product had a boolean if it was either spicy or vegetarian.
- The file with the branches was just a text file with all the addresses of the branches.
- Different kinds of ingredients can just be imported into the new database.

Normalisation

Normalisation is a guideline to avoid data redundancy and avoid dataloss. If we take a look the orders work, we can apply some normalisation to simplify this process. An Order exists of multiple parts:

1NF
<u>Order ID</u> PK
Branch
Address
Customer

Phone number
Email address
Notes
Order Placed At
Order Delivered At
Order line
Coupons

2NF

ID PK
BranchId FK
AddressId FK
CustomerId FK
Phone
Email
Notes
PlacedAt
DeliveredAt
OrderLine
Coupons