

05/07/2018

Oracle database

data project

written BY:

Dylan cattelan,

cedric montes,

anthony luque,

anais banos

cesi exia a3

Table of Contents

[Oracle database 2](#_Toc518595221)

[MCD – Modèle de conception de données 2](#_Toc518595222)

[Color, Variant, Texture 3](#_Toc518595223)

[CandyCost / Candy 3](#_Toc518595224)

[Packaging, Palette, Cardboard 3](#_Toc518595225)

[CandyReferences 4](#_Toc518595226)

[Orders 4](#_Toc518595227)

[Country 4](#_Toc518595228)

[Stock 4](#_Toc518595229)

[Machine 4](#_Toc518595230)

[MLD – modèle logique de données 5](#_Toc518595231)

[MPD – Modèle physique de données 5](#_Toc518595232)

[Data Generator 6](#_Toc518595233)

[UML PYTHON 6](#_Toc518595234)

[Rights and Privileges 12](#_Toc518595235)

iNTRODUCTION

To introduce this report, I would like to explain the situation.

During this week, we have to replicate the operation of a candy factory. Thereby, we had to conceive, develop and deploy an Oracle DB, a MongoDB + an ETL, and a Data Generator.

We split our work in two main parts, and two small ones:

* Oracle DB
* Data Generator

+ ETL / MongoDB / Reporting Stats

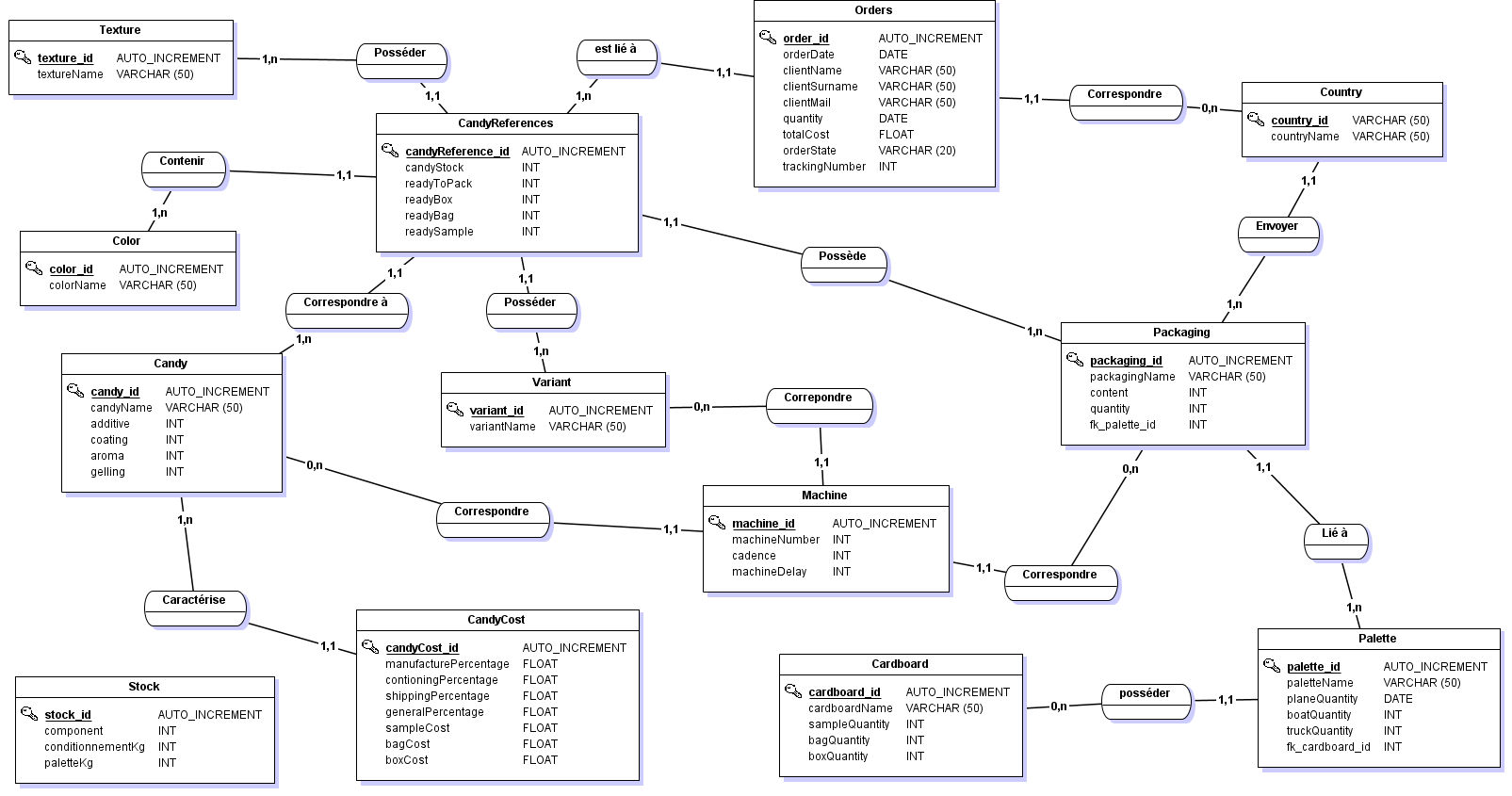
+ RO / Graphes

The Oracle DB concerns the internal operation of the factory, and the Data Generator simulates the external flows of the company, such as orders or references creation. Concerning the other parts, they will be described in another document.

# Oracle database

## MCD – Modèle de conception de données

This schema describes how we imagined our database, there are different links and associations between different tables. We used the MERISE conventions to do that.



As we can see above in our database design schema, we created a relational Database.

It contains the following 13 tables:

* Color
* Variant
* Texture
* Stock
* Cardboard
* Palette
* Packaging
* Machine
* CandyCost
* Candy
* CandyReferences
* Orders
* Country

To explain them, we will regroup some tables according to their characteristics.

### Color, Variant, Texture

We chose to create specific tables for these data, to anticipate their evolution. For example, if the company wants to add another color, this new addition will be managed in a natural way with a new ID in the corresponding table.

### CandyCost / Candy

In this case, we decided to split these data in two tables because of the different sources of the data. CandyCost group all the Commercial Data concerning the different costs, like Manufacturing or Box price. The second one concerns the composition of the candy with the different quantities of each components.

### Packaging, Palette, Cardboard

These tables are different than the others, because we are using a recursive method to populate the Packaging Table.

Because, for instance, a Box contains 25 Candies, we decided to implement a foreign key to the same table, in order to say “Boxes are filled with candies, and a candy is worth 1 candy”. This way, no need to split tables.

But we had an issue with Palettes and Cardboards, because Cardboards are filled with either Samples, Boxes or Bags, so we had to handle this on a different table.

Same thing with Palettes, they are filled with Cardboards, but it depends on what kind of cardboard ( Samples, Boxes, Bags ).

This way, we are optimizing the tables because it only requires to add a shipping method on the Packaging method, or even another candy bag to the same table to directly link them to other methods.

### CandyReferences

This table will group all the unique candy references, we chose to add an ID in this table to easy up the use and lighten the different documents.

Thus, using the Python Generator, and before adding a new reference, the generator will check if there’s an already existing reference instead of copying it.

This table handles every variant, texture, color and packaging possible for every candy.

### Orders

This table will be used to store order informations like the order Datetime or the client name. It can be important to have a specific table. It makes it easier when we search information about a specific order. It prevents us to join some tables, in our case all information is stored in an optimal table.

### Country

The Country Table groups the different possibilities of country shipping. And we can see the table is link with the packaging table we explained above. Thanks to the packaging\_id, we can easily find the shipping method for every country.

### Stock

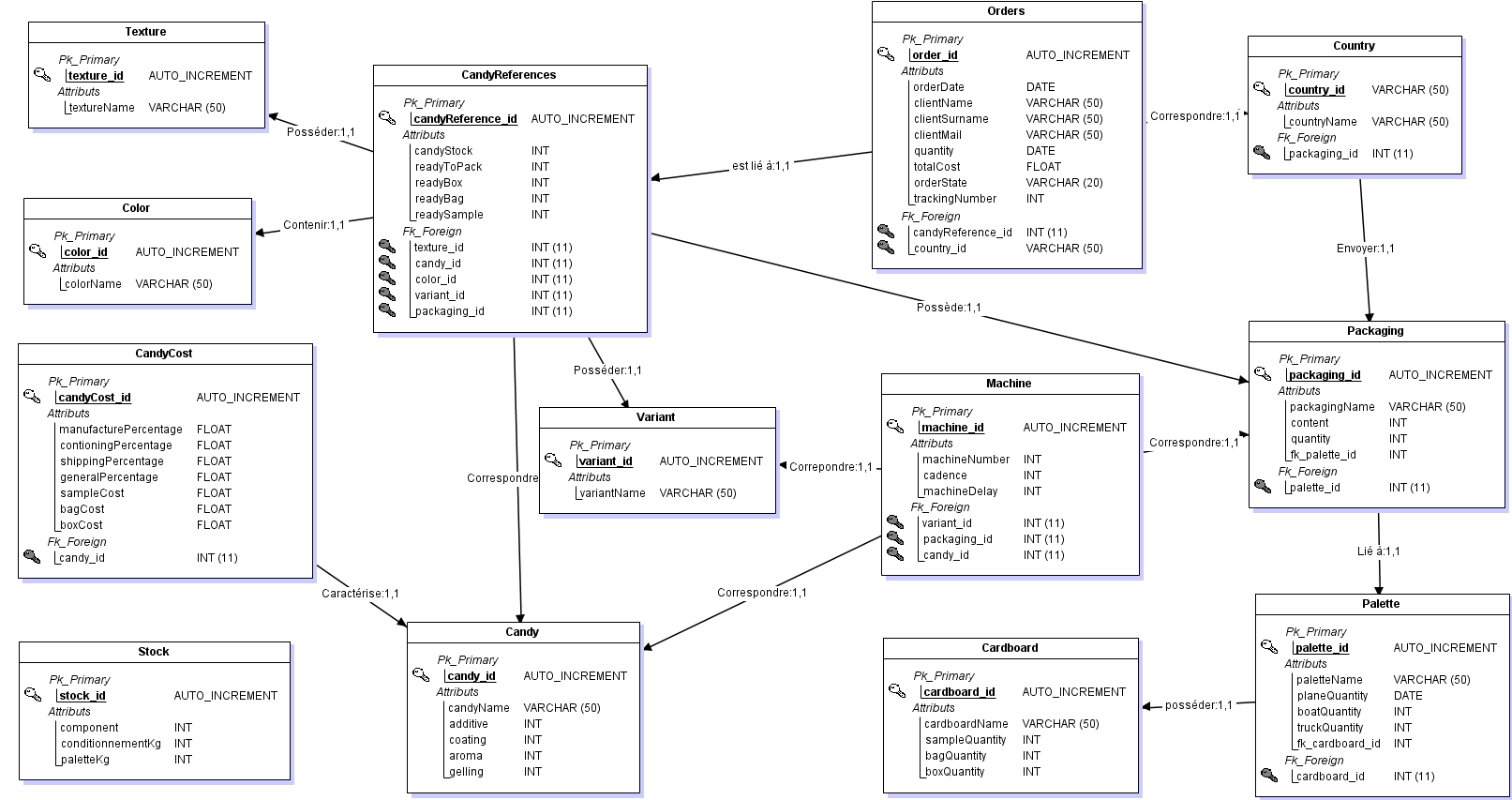
Stock is the only table without connection, because information stored in it concerns raw material supply and is managed by the generator.

### Machine

We group Manufacturing Machine and Conditioning machine in Machine table, we use foreign keys to differentiate the two of them. In the first case, if the fk\_variant\_id is not null, it means it is a manufacturing machine, on the opposite, if the fk\_packaging\_id is not null and the fk\_variant\_id is null ( same result, but in a different way ) we can say we are still talking about a conditioning Machine.

## MLD – modèle logique de données

## 



This is a logical schema, that means there is no association table, but there are the 13 main tables with the different primary and foreign keys which shows us how they are linked to the other one.

This is the same explanation as MCD.

## MPD – Modèle physique de données

Cf the « reqCreateTable.sql » file.

That is totally different from the previous explanations, because we built it according to the programming language. This schema represents the previous one with the adapted syntax.

As we can see, there are different parts in this SQL script.

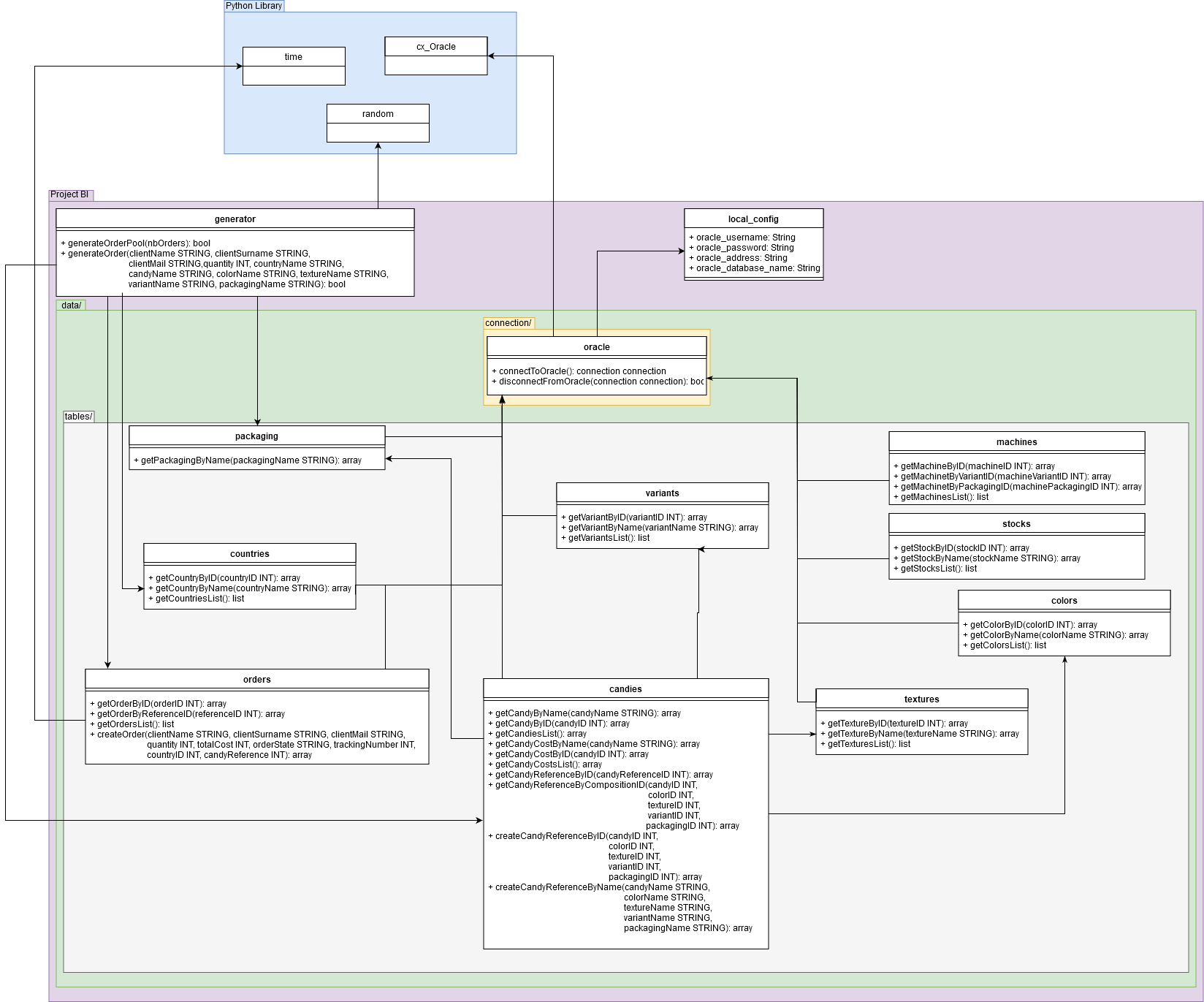
On the one hand, we have the creation part, it is used to create all the necessary tables for the database to function properly.

On the other hand, we have commands to insert data into the previous tables, it is used to insert a dataset into the database, and it is called a databank.

This databank will be used to do our calculations and our data generations.

# Data Generator

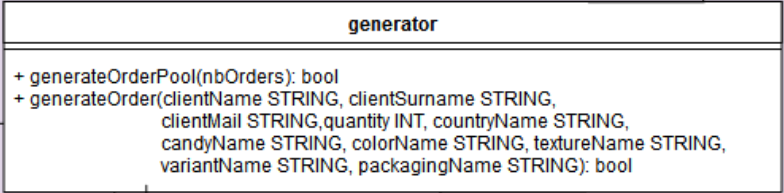
## UML PYTHON



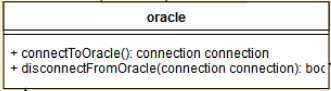
This UML show you the interconnection between the project’s Python classes. As you can see, the folder is properly organized, so we can easily navigate and find the desired class.

For instance, to get the candy list, you’ll have to call the method “Project BI.data.tables.candies.py”.getCandiesList(). This way, you directly know what you are looking for and where to look at.

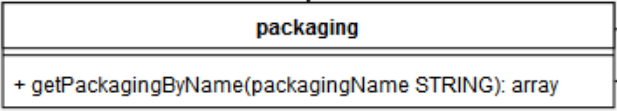
Moreover, we separated the connection part from the classes, so we now have a file called oracle.py where lies all the connection part. And, for security reasons, the credentials aren’t directly in this file. They are at the root of the folder, in a file called “local\_config.py”, where you’ll have to set the variables with your computer’s Oracle credentials. This file has been added to .gitignore, this way, there are no possibilities for anyone to get your credentials. There is a “local\_config\_example.py” file to show you how to do this, plus the instructions on the README.MD of the project.



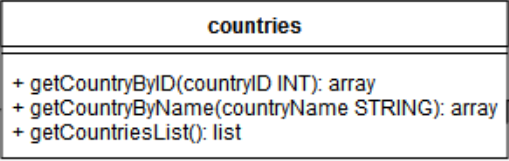
* generateOrderPool(nbOrders INT) -> a method to create a batch of random orders defined by the parameter sent
* generateOrder( [ … ] ) -> a method to create a specific order according to parameters written



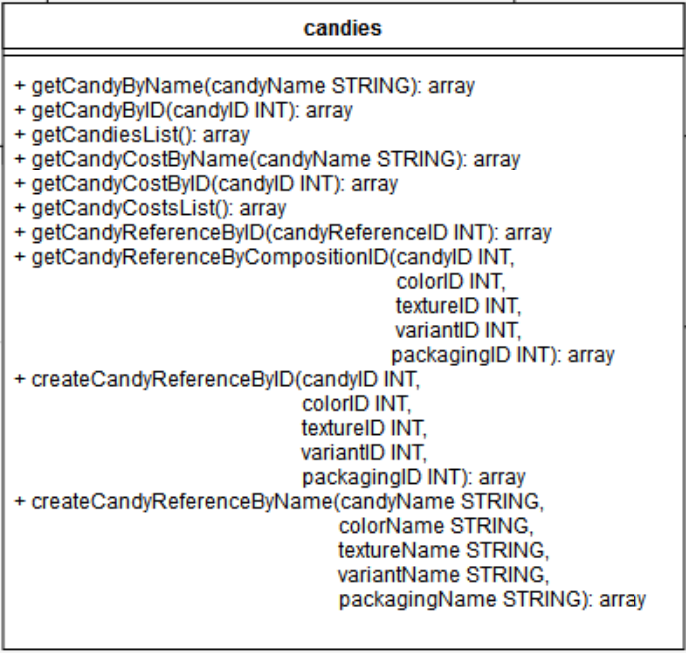
* connectToOracle() -> used to enable the connection with the database, returns a connection object used to query the database with
* disconnectFromOracle(connection connection) -> a method to terminate the connection with the database ( the connection parameter is used to terminate it )



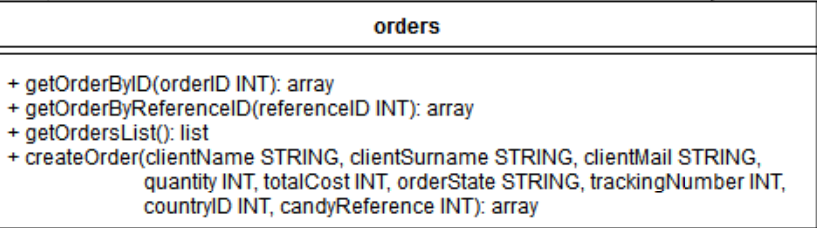
* getPackagingByName(packagingName STRING ) -> used to get the packaging ID according to the packaging name sent as a parameter



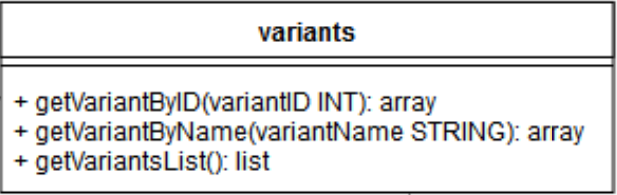
* getCountryByID(countryID INT) -> a method that returns the country name + the shipping method based on a given ID
* getCountryByName ( countryName STRING ) -> same as the previous method, but using the country Name directly
* getCountriesList() -> method that returns a list of all the countries + shipping method



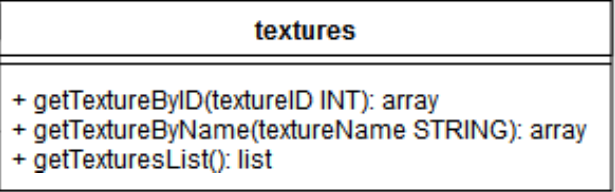
* getCandyByName(candyName STRING) -> returns the candy description for a given Name
* getCandyByID(candyID INT) -> same as the previous one, but using an ID instead
* getCandiesList() -> returns a list of all the candies
* getCandyCostByName(candyName STRING) -> returns the candy cost for a given Name
* getCandyCostByID(candyID INT ) -> same as the previous one, but using an ID instead
* getCandyCostsList() -> returns a list of all the candies with the costs associated
* getCandyReferenceByID(candyReferenceID INT) -> finds a reference using an ID ( if exists )
* getCandyReferenceByCompositifionID([…]) -> same as before but using the compositions ID
* createCandyReferenceByID([…]) -> checks if the reference already exists, and if not, creates a reference using composition ID
* createCandyReferenceByName([…]) -> same as before, but using Names directly

.

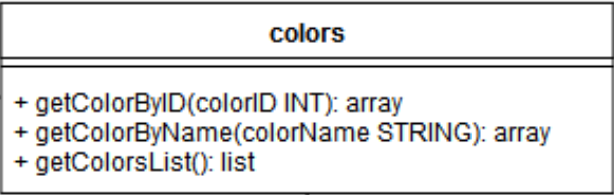
* getOrderByID(orderID INT) -> returns the order associated to the ID given
* getOrderByReferenceID(referenceID INT) -> returns the orders with the referenceID associated to
* getOrdersList() -> displays a list of all the orders
* createOrder( [ … ] ) -> create an order with all the informations given in parameters



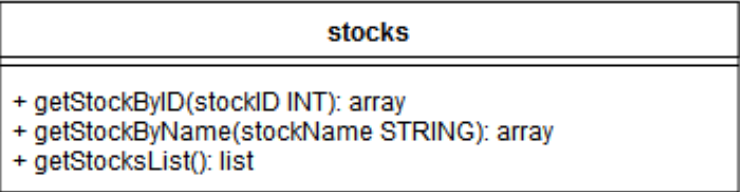
* getVariantByID(variantID INT) -> a method that returns the variant name
* getVariantByName (variantName STRING ) -> same as the previous method, but using the variant Name directly
* getVariantsList() -> method that returns a list of all the variants available



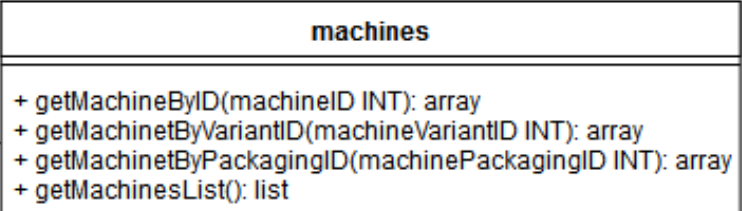
* getTexturetByID(textureID INT) -> a method that returns the texture name
* getTextureByName (textureName STRING ) -> same as the previous method, but using the texture Name directly
* getTexturesList() -> method that returns a list of all the textures available



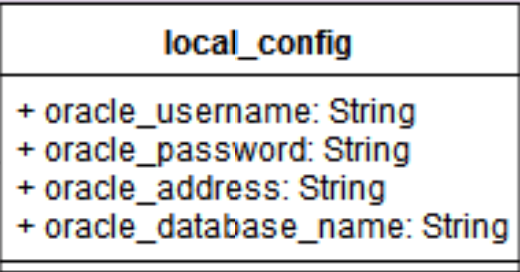
* getColorByID(colorID INT) -> a method that returns the color name
* getColorByName (colorName STRING ) -> same as the previous method, but using the color Name directly
* getColorsList() -> method that returns a list of all the colors available



* getStockByID(stockID INT) -> a method that returns the stocks name
* getStockByName (stockName STRING ) -> same as the previous method, but using the stocks Name directly
* getStocksList() -> method that returns a list of all the stocks available

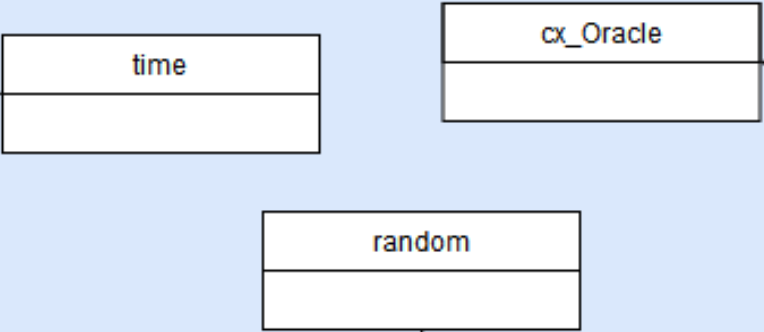


* getMachineByID(machineID INT) -> a method that returns the machine infos
* getMachineByVariantID (machineVariantID INT ) -> same as the previous method, but using the variant Name directly
* getMachineByPackagingID (machinePackagingID INT ) -> same as the previous method, but using the packaging Name directly
* getMachinesList() -> method that returns a list of all the stocks available



( Protected file, it has been added to the .gitignore for security reasons )

* oracle\_username -> a variable used to get the oracle database username
* oracle\_password -> a variable used to get the oracle database password
* oracle\_address -> a variable used to get the oracle database address
* oracle\_database\_name -> a variable used to get the name of the oracle database



* Time -> a Python library to get current time and do things with time and date
* Random -> a Python library to add randomness ( used to generate orders )
* Cx\_Oracle -> a Python library to connect scripts to an Oracle Database

# Rights and Privileges

Concerning the Oracle database, we decided to build 2 users, for obvious security reasons.  
The first one is called “generator”, it will be able to SELECT any tables and to INSERT and UPDATE only the following tables:

- Orders

- CandyReferences

- Stock

The generator only needs to GET data, and the only possibilities it’ll deal with the 3 previous tables is when there’s an order. It does not need any other rights.

Concerning the talend user, he will only have the SELECT command available on every tables of the Oracle database, no modifications are directly allowed so he won’t have any others rights.