Groupe : Rachid, Phichet, Camille et Joséphine

Contexte du brief

L'objectif de ce brief était d'intégrer une société de e-commerce en tant que développeur IA et de les aider à la refonte de leur base de données. À partir d'une base de données existante, nous devions l'implémenter dans MySQL par le biais des langages Python et SQL. Nous devions également utiliser différentes technologies (SQLite, mysql-connector...) et concevoir et optimiser notre base de données au sein d'un éditeur de code Python (Visual Studio Code ou Pycharm).

La première étape a consisté à analyser et comprendre la donnée pour l'organiser et l'agencer de la manière la plus claire et optimale possible. Nous avons ainsi procédé à la réalisation d'un modèle de classe (MPD) selon les règles de cardinalité et avons créé des foreign keys et des primary keys selon le rôle de chaque table. L'essentiel pour nous était dans cette étape de rendre lisible et compréhensible le modèle de la base de données pour le client qui pourrait être amené à le consulter.

Une des étapes importantes a été aussi notre capacité à chercher dans la documentation des solutions aux difficultés que nous pouvions rencontrer. Il s'agissait ainsi pour nous de nous familiariser avec des langages et logiciels jusque-là peu connus et de les combiner les uns avec les autres pour gagner du temps et de l'efficacité (Pandas, Pymysql...).

D'autre part, la dernière étape a consisté en la création de notre base de données sur des langages adaptés. Nous avons ensuite nettoyé les données de certains fichiers CSV qui étaient mis à notre disposition par le client. Nous avons enfin intégré l'ensemble de ces données dans les tables conçues spécifiquement pour elles.

Les compétences qui étaient ciblées pour ce brief étaient:

-Conception et optimisation de base de données relationnelles.

-Nettoyage des données

-Intégration des données dans un modèle de base de données relationnelles à partir de fichiers sources CSV.

- Réaliser des requêtes SQL dans un environnement python conçues en fonction des besoins du client. (Panier moyen, satisfaction moyenne…)

Technologies utilisées

Pour ce brief, nous avons pu expérimenter deux façons de procéder. En effet, nous avons utilisé soit MySQL, soit SQLite pour la conception de la base de données, de la création de ses tables au requêtage de celle-ci.

Pour la partie MySQL, elle s'est faite via l'éditeur de code PyCharm tandis que la partie SQLite a été faite par le biais de Visual Studio Code. Chacun d'entre elles présentait quelques avantages mais aussi quelques inconvénients.

MySQL

La partie réalisée sur MySQL a été réalisée sur PyCharm grâce à l'importation de mysql-connector. La rédaction du code a été répartie en 3 fichiers py : un premier fichier pour créer les tables, un deuxième pour importer les données dans la base de données et un troisième pour les requêtes SQL.

Lors de nos recherches pour importer les données, nous avons réalisé que la fonction to_sql de Pandas permettait d'importer de la donnée depuis un dataframe en table SQL en créant la table en même temps. Pour utiliser la fonction to_sql, nous avons dû installer le package sqlalchemy pour faire le lien entre le dataframe et le sql. Le premier fichier de création de tables est donc devenu redondant et inutile. La rédaction du code grâce à MySQL s'est déroulée sans difficultés.

SQLite

La partie SQLite a été assez intuitive à mettre en place. Le fait de pouvoir gérer la totalité de la base de données au sein d'un fichier de format db était intéressant car les modifications se faisaient facilement là-dessus. Ne pas devoir se baser sur un modèle client/serveur semblait rendre les choses plus rapides à réaliser grâce à la modification de ce fichier.

La rédaction du code à l'aide de SQLite s'est faite sur un notebook Jupyter (donc sur un fichier au format ipynb). Son utilisation a été très intuitive et a permis de bien fragmenter et commenter le code. De ce fait, trois fichiers ipynb ont été créés : un fichier pour la création des tables, un fichier pour l'intégration des données dans ces tables et un fichier pour les requêtes. Il fallait donc mettre en lien ces fichiers à l'aide d'un import et il fallait donc les convertir au format py (les deux premiers fichiers) pour cela.

SQLite, sur ce projet avec cette base de données, n'a pas réellement mis en valeur d'inconvénients car les données restaient assez légères à manipuler. Peut-être qu'une base de données plus conséquente aurait posé problème.

Pour le reste, que ce soit MySQL ou SQLite, l'écriture du code restait assez similaire, notamment en important la bibliothèque Pandas et en requêtant à l'aide de SQL.

Autres outils

Dans la lecture des fichiers CSV, une extension intéressante a été utilisée sur VSCode : Excel Viewer. Cela nous a permis de mieux lire nos tableaux de données et de repérer les incohérences que l'on a pu relever.

MPD présenté

Lors de la première partie de l'exercice, nous avons réalisé un MPD (Modèle Physique des Données) à partir du contexte du brief. Nous avons pensé à un modèle comprenant 14 tables, réparties en plusieurs catégories (code couleur).

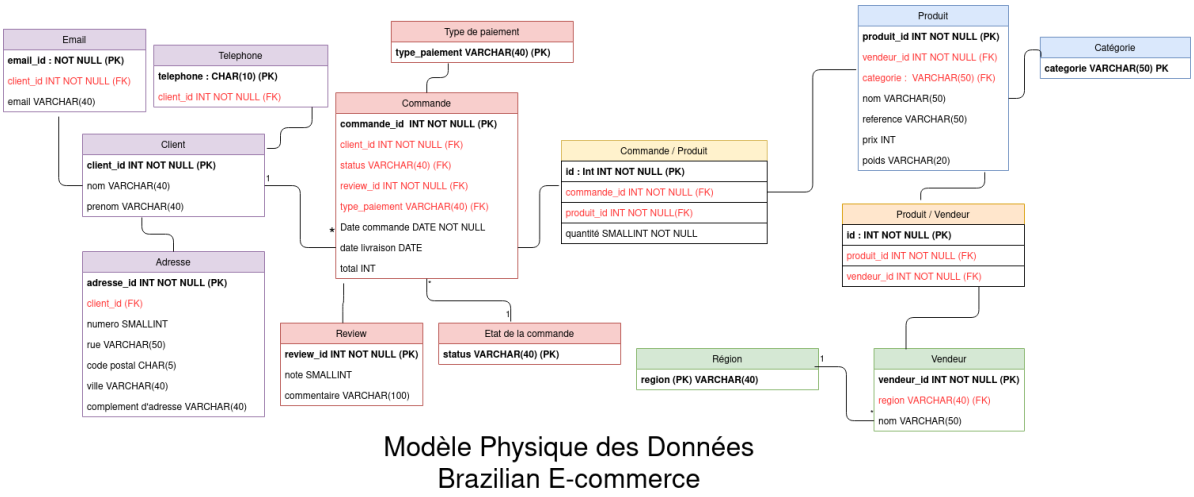

Image 1. Modèle Physique des Données

La table Client, autour de laquelle s'ajoutent les tables Téléphone, Adresse et Email. La Table Commande, connectée à la table Client et reliée aux tables Type de paiement, Review et Etat de la commande. Une table de jointure entre Commande et Produit. La table Produit, reliée à la table Catégorie de produit. Une table de jointure entre Produit et Vendeur. Une table Vendeur, reliée à une table Région

Lors du partage des données de Kaggle, nous avons réalisé que notre premier MPD avait trop de tables par rapport aux données fournies. Nous avons donc réfléchi à un nouveau MPD.

Image 2. MPD réalisé à partir des données Kaggle

Ce nouveau MDP comporte 8 tables : Order, Order_payment, Order_review, Customer, Order_items, Geolocation, Sellers et Products. Il nous a servi par la suite pour créer la base de données SQL.

Création et intégration de la BDD étape par étape

Nous allons vous exposer la création et l'intégration de la BDD sur MySQL et SQLite

MySQL

Tout d'abord nous avons créé la database

```
mycursor = mydb.cursor()
mycursor.execute("CREATE DATABASE e_commerce")
```

Ensuite en se connectant à la database nous avons créé les tables. Nous présenterons le code pour la table Customers.

Nous avons d'abord transformé le fichier csv en dataframe Pandas.

```
customers = pd.read_csv("data/olist_customers_dataset.csv")
customers_df = pd.DataFrame(customers, columns=
['customer_id','customer_unique_id','customer_zip_code_prefix', 'customer_city',
'customer_state'])
```

Nous avons ensuite nettoyé le dataframe, si des lignes étaient en double. Cette étape n'a pas été faite pour toutes les tables, seulement pour celles qui présentaient des anomalies.

```
# we get rid of any duplicates on the customer_unique_id column
customers_df.drop_duplicates(subset=['customer_unique_id'], inplace= True)
print(customers_df.shape)
```

Nous avons ensuite importé le dataframe dans une table SQL.

```
# import the dataframe into sql table
customers_df.to_sql('Customers', engine, if_exists='replace', index= False)
```

Nous avons réalisé ces opérations pour l'ensemble des tables.

SQLite

Dans un premier temps, on importe le module SQLite3 et on crée la BDD grâce à la méthode .connect()

```
import sqlite3

conn = sqlite3.connect('e_commerce.db')
c = conn.cursor()
```

On intègre les différentes tables à notre BDD. Voici un exemple avec la table Products :

```
c.execute('''CREATE TABLE IF NOT EXISTS Products (
    product_id text NOT NULL PRIMARY KEY,
    product_category_name text,
    product_name_lenght integer,
    product_description_lenght integer,
    product_photos_qty integer,
    product_weight_g integer,
    product_length_cm integer,
    product_height_cm integer,
    product_width_cm integer
    )''')
```

Après avoir créé nos tables, on procède à l'importation de nos données et au nettoyage des fichiers dans lesquels nos données se trouvent (ici, dans les fichiers CSV)

Première étape : nouveau fichier donc on remet les imports nécessaires et la connexion à SQLite

```
import sqlite3
import pandas as pd
import tables_creation

conn = sqlite3.connect('e_commerce.db')
c = conn.cursor()
```

Deuxième étape : on importe nos données dans les tables préalablement créées (ici, pour la table Products)

```
"""Importation des données de Products dans la table Products"""

products = pd.read_csv('olist_products_dataset.csv')
products.to_sql('Products', conn, if_exists='replace', index = False)
```

Si nécessaire, si la donnée doit être nettoyée, on doit passer par la DataFrame fournie par Pandas

```
"""Importation des données de Geolocation dans la table Geolocation"""

geolocation = pd.read_csv('olist_geolocation_dataset.csv')

""" Suppression des doublons de la colonne geolocation_zip_code_prefix """

geolocation_df = pd.DataFrame(geolocation)
geolocation.drop_duplicates(subset=
['geolocation_zip_code_prefix']).to_sql('Geolocation', conn, index = False,
if_exists='replace')
```

Choix à justifier si évolution des variables

La base de données initiale a mis en évidence quelques incohérences dans le cadre de notre brief. Il nous fallait de ce fait nettoyer cette base de données avant de l'intégrer la plus propre possible dans nos tables préalablement créées.
En étudiant les fichiers CSV, nous avons donc remarqué plusieurs choses.

Table Geolocation

Dans un premier temps, dans la table Geolocation, nous avions à l'origine 5 colonnes (geolocation_zip_code_prefix, geolocation_lat, geolocation_lng, geolocation_city, geolocation_state). Dans le cadre du brief, deux choses nous ont interpellés.

La première est la présence des colonnes geolocation_lat et geolocation_lng. Elles nous semblaient inutiles donc nous avons fait le choix de les supprimer de la base de données. Elles ne se retrouvaient dans aucune autre table.La seule chose qui pouvait nous ennuyer était qu'elles permettaient de différencier les codes postaux qui étaient relevés dans cette table. Or, dans les autres tables, seul le code postal subsiste donc ne pas garder ces deux colonnes nous semblait logique.

Cela nous amène au second point de cette table qui est la suppression des doublons de la colonne geolocation_zip_code_prefix. En effet, à l'origine, nous retrouvons plus de 1 million de codes postaux pour un peu plus de 19 000 valeurs uniques. Nous n'avions pas besoin d'autant de codes postaux. De ce fait, afin de faire le lien avec les autres tables possédant ces codes postaux, nous avons supprimé tous les doublons. Le code postal va localiser une zone assez restreinte donc un seul code postal peut nous permettre de localiser relativement facilement un client ou un vendeur.

Table Order_reviews

Dans notre quête des incohérences de la donnée, nous avons relevé une dernière chose dans la table Order_reviews. En effet, nous avions certaines reviews qui étaient doublées en analysant la colonne review_id. Ces reviews avaient pour particularité d'avoir le même commentaire mais pour un order_id différent, ce qui nous semblait curieux. Pourtant, la review était postée exactement à la même heure pour chaque doublon. Nous avons donc supprimé les doublons de cette table.

Présentation du code SQL pour chaque indicateur

Nombre de clients total

```
SELECT COUNT(*)
FROM Customers
```

```
> 96096
```

Nombre de produits total

```
SELECT COUNT(product_id)
FROM Products
```

```
> 32951
```

Nombre de produits par catégorie

```
SELECT product_category_name, COUNT(product_id)
FROM Products
GROUP BY product_category_name
ORDER BY COUNT(product_id) DESC
```

```
> ('cama_mesa_banho', 3029), ('esporte_lazer', 2867), ('moveis_decoracao', 2657),
('beleza_saude', 2444), ('utilidades_domesticas', 2335), ('automotivo', 1900),
('informatica_acessorios', 1639), ('brinquedos', 1411), ('relogios_presentes',
1329), ('telefonia', 1134), ('bebes', 919), ('perfumaria', 868),
('fashion_bolsas_e_acessorios', 849), ('papelaria', 849), ('cool_stuff', 789),
('ferramentas_jardim', 753), ('pet_shop', 719), (None, 610), ('eletronicos',
517), ('construcao_ferramentas_construcao', 400), ('eletrodomesticos', 370),
('malas_acessorios', 349), ('consoles_games', 317), ('moveis_escritorio', 309),
('instrumentos_musicais', 289), ('eletroportateis', 231), ('casa_construcao',
225), ('livros_interesse_geral', 216), ('fashion_calcados', 173), ('moveis_sala',
156), ('climatizacao', 124), ('livros_tecnicos', 123), ('telefonia_fixa', 116),
('casa_conforto', 111), ('market_place', 104), ('alimentos_bebidas', 104),
('fashion_roupa_masculina', 95),
('moveis_cozinha_area_de_servico_jantar_e_jardim', 94),
('sinalizacao_e_seguranca', 93), ('construcao_ferramentas_seguranca', 91),
('eletrodomesticos_2', 90), ('construcao_ferramentas_jardim', 88), ('alimentos',
82), ('bebidas', 81), ('construcao_ferramentas_iluminacao', 78),
('agro_industria_e_comercio', 74), ('industria_comercio_e_negocios', 68),
('artigos_de_natal', 65), ('audio', 58), ('artes', 55),
('fashion_underwear_e_moda_praia', 53), ('dvds_blu_ray', 48), ('moveis_quarto',
45), ('construcao_ferramentas_ferramentas', 39), ('livros_importados', 31),
('portateis_casa_forno_e_cafe', 31), ('pcs', 30), ('cine_foto', 28),
('fashion_roupa_feminina', 27), ('musica', 27), ('artigos_de_festas', 26),
('fashion_esporte', 19), ('artes_e_artesanato', 19), ('flores', 14),
('fraldas_higiene', 12), ('portateis_cozinha_e_preparadores_de_alimentos', 10),
('la_cuisine', 10), ('moveis_colchao_e_estofado', 10),
('tablets_impressao_imagem', 9), ('casa_conforto_2', 5),
('fashion_roupa_infanto_juvenil', 5), ('pc_gamer', 3), ('seguros_e_servicos', 2),
('cds_dvds_musicais', 1)
```

Nombre de commandes total

```
SELECT COUNT(DISTINCT order_id)
FROM Orders
```

```
> 99441
```

Nombre de commandes selon leurs états (en cours de livraison...)

```
SELECT order_status, COUNT(DISTINCT order_id)
FROM Orders
GROUP BY order_status
```

```
> ('approved', 2), ('canceled', 625), ('created', 5), ('delivered', 96478),
('invoiced', 314), ('processing', 301), ('shipped', 1107), ('unavailable', 609)
```

Nombre de commandes par mois

```
SELECT EXTRACT(MONTH FROM order_purchase_timestamp) AS Month,
COUNT(DISTINCT order_id)
FROM Orders
GROUP BY Month
```

```
> (1, 8069), (2, 8508), (3, 9893), (4, 9343), (5, 10573), (6, 9412), (7, 10318),
(8, 10843), (9, 4305), (10, 4959), (11, 7544), (12, 5674)
```

Prix moyen d'une commande

```
WITH Total AS
            (
            SELECT DISTINCT(order_id), SUM(payment_value) AS pv
            FROM Order_payments
            GROUP BY order_id
            )
            SELECT AVG(pv)
            FROM Total
```

```
> 160.9902666934708
```

Score de satisfaction moyen

```
SELECT AVG(review_score)
FROM Order_reviews
```

```
> 4.0709
```

Nombre de vendeurs

```
SELECT COUNT(DISTINCT seller_id)
FROM Sellers
```

```
> 3095
```

Nombre de vendeurs par région

```
SELECT seller_state, COUNT(DISTINCT seller_id)
FROM Sellers
GROUP BY seller_state
```

```
> ('AC', 1), ('AM', 1), ('BA', 19), ('CE', 13), ('DF', 30), ('ES', 23), ('GO',
40), ('MA', 1), ('MG', 244), ('MS', 5), ('MT', 4), ('PA', 1), ('PB', 6), ('PE',
9), ('PI', 1), ('PR', 349), ('RJ', 171), ('RN', 5), ('RO', 2), ('RS', 129),
('SC', 190), ('SE', 2), ('SP', 1849)
```

REQUETES PLUS AVANCEES

Quantité de produits vendus par catégorie

```
SELECT p.product_category_name as Categorie_produit, COUNT(o.order_id) as
Nombre_commandes
FROM Orders_items as o
INNER JOIN Products as p
ON o.product_id = p.product_id
GROUP BY p.product_category_name
ORDER BY Nombre_commandes DESC
```

```
> Categorie_produit Nombre_commandes
0   cama_mesa_banho 11115
1   beleza_saude    9670
2   esporte_lazer   8641
3   moveis_decoracao    8334
4   informatica_acessorios  7827
... ... ...
69  la_cuisine  14
70  cds_dvds_musicais   14
71  pc_gamer    9
72  fashion_roupa_infanto_juvenil   8
73  seguros_e_servicos  2
74 rows × 2 columns
```

Nombre de commandes par jour

Pour cette requete, nous avons choisi 2 approches :

- Nombre de commandes par jour dans le mois (de 1 à 31)

```
SELECT EXTRACT(DAY FROM order_purchase_timestamp) AS Day,
COUNT(DISTINCT order_id)
FROM Orders
GROUP BY Day
```

```
> [(1, 3101), (2, 3213), (3, 3283), (4, 3483), (5, 3445), (6, 3468), (7, 3363),
(8, 3326), (9, 3271), (10, 3168), (11, 3308), (12, 3202), (13, 3277), (14, 3387),
(15, 3524), (16, 3581), (17, 3200), (18, 3430), (19, 3364), (20, 3261), (21,
3116), (22, 3181), (23, 3128), (24, 3877), (25, 3290), (26, 3290), (27, 3122),
(28, 3011), (29, 2557), (30, 2534), (31, 1680)]
```

- Nombre de commandes par jour de la semaine (1 étant dimanche, jusque 7 samedi)

```
SELECT DAYOFWEEK(order_purchase_timestamp) AS Day_of_week,
COUNT(DISTINCT order_id)
FROM Orders
GROUP BY Day_of_week
```

```
> [(1, 11960), (2, 16196), (3, 15963), (4, 15552), (5, 14761), (6, 14122), (7,
10887)]
```

Durée moyenne entre la commande et la livraison (exprimée en jours)

```
SELECT AVG(DATEDIFF(order_delivered_customer_date, order_purchase_timestamp))
FROM Orders
```

```
> 12.4973
```

Nombre de commandes par ville

```
SELECT s.seller_city as Ville_vendeur, COUNT(o.order_id) as Nombre_commandes
FROM Orders_items as o
INNER JOIN Sellers as s
ON o.seller_id = s.seller_id
GROUP BY Ville_vendeur
ORDER BY Nombre_commandes DESC
```

```
> Ville_vendeur Nombre_commandes
0   sao paulo   27983
1   ibitinga    7750
2   curitiba    3016
3   santo andre 2964
4   belo horizonte  2593
... ... ...
606 araquari    1
607 angra dos reis rj   1
608 aguas claras df 1
609 abadia de goias 1
610 04482255    1
611 rows × 2 columns
```

Prix minimum des commandes

```
WITH Total AS
            (
            SELECT DISTINCT(order_id), SUM(payment_value) AS pv
            FROM Order_payments
            GROUP BY order_id
            )
            SELECT MIN(pv)
            FROM Total
```

```
> 0.0
```

Prix maximum des commandes

```
WITH Total AS
            (
            SELECT DISTINCT(order_id), SUM(payment_value) AS pv
            FROM Order_payments
            GROUP BY order_id
            )
            SELECT MAX(pv)
            FROM Total
```

```
> 13664.08
```

Temps moyen d'une livraison par mois (requête faite avec SQLite)

```
SELECT strftime('%m',order_purchase_timestamp) as Month,
AVG(julianday(order_delivered_customer_date) -
julianday(order_purchase_timestamp)) as duree_livraison_par_mois_en_jours
FROM Orders
GROUP BY Month
```

```
>
Month   duree_livraison_par_mois_en_jours
0   01  13.943059
1   02  16.186325
2   03  15.408157
3   04  12.364333
4   05  11.385740
5   06  10.180754
6   07  9.975710
7   08  9.090552
8   09  11.861356
9   10  12.295812
10  11  15.161132
11  12  15.392978
```