

Rapport projet BDD e-commerce

Sommaire

Rapport projet BDD e-commerce

- Sommaire
- Organisation du projet
- Technologies
- Présentation du MCD
- Création de la DataBase
- Gestion des erreurs
- Présentation des requêtes SQL
 - 1) Fonction GetNB
 - 2) Fonction getAvgNB
 - 3) Requête 'get orders by state'
 - 4) Requête 'get orders by month'
 - 5) Requête 'average orders score'
 - 6) Requête 'sellers by state'
 - 7) Fonction newProduct

Organisation du projet

Pour l'organisation du projet, nous avons utilisé l'outil **Trello** afin de décomposer au mieux les différentes tâches du projet. Nous avons adopté une méthodologie en 6 étapes :

- **To-Do** : rassemble l'ensemble des tâches restantes à réaliser
- **Doing** : rassemble les tâches que les membres du projet se sont assignés
- **Help** : compose des tâches sur lesquelles les membres du projet ont besoin d'aide
- **Done** : correspond aux tâches que les membres du groupe ont finalisés
- **Merge** : recense l'ensemble des codes qui ont pu être merge via Git

Pour collaborer efficacement sur ce projet, nous avons créé un répertoire sur **Github** via lequel chacun des membres a pu travailler sur sa branche respective.

Technologies





Présentation du MCD

*Lien vers le MCD : <https://github.com/MistSF/Simplon-Agence-KJK/blob/main/MCD.pdf>

Création de la DataBase

Lien vers le répertoire Github du projet : <https://github.com/MistSF/Simplon-Agence-KJK>*

Pour la création des BDD, nous nous sommes appuyés sur les données de 8 fichiers csv au total. Chaque fichier concentre l'ensemble des informations pour chaque table de la base de données, comme représenté dans la présentation du MPD ci-dessus. Dans un autre fichier, *dictionnary.py*, nous créons les requêtes pour la création des tables avec les types de ces valeurs et les clés primaires/ clés étrangères de chacune des tables : *Code de création d'une table*

Nous récupérons l'ensemble des informations contenues dans ces fichiers csv grâce à une fonction d'import des données contenue dans un fichier *initial.py* :

```
def createDatabase(cursor, database) :
    request = "CREATE DATABASE {}".format(database)
    try :
        cursor.execute(request)
    except mysql.connector.Error as err :
        print("creation failed : {}".format(err))

request = "USE {}".format(database)
try :
    cursor.execute(request)
except mysql.connector.Error as err :
    print("connection failed : {}".format(err))
```

Gestion des erreurs

Pour traiter connaître les erreurs présentes dans chaque table, nous créons la fonction `saveError`, qui enregistre l'ensemble des erreurs rencontrées pour chacune des bases dans plusieurs fichiers `.txt` :

```
def saveError(name, request, value) :  
    f = open("./Error/{}.txt".format(name), "a")  
    f.write("{} : {}\n{}\n\n".format(name, request, value))  
    f.close()
```

Lors de l'exécution du code, nous avons rencontré un certain nombre d'erreurs :

- Certains types n'étaient pas adaptés aux données importées (ex : limitation VARCHAR trop faible par rapport à la taille de la donnée importée) -> Nous avons ajusté les types directement depuis le fichier `actionDatase.py`
- La présence de nombreux doublons au sein des clés primaires -> Nous avons utilisé la fonction `drop_duplicates` de pandas pour ne garder que la première occurrence de l'id de la clé primaire dans la table
- Quelques problèmes de syntaxe SQL et Python, qui ont pu être rapidement réglés (ex : l'oubli de la méthode `fetchall()` pour certaines fonctions qui renvoyait un `None` lorsque nous essayons d'exécuter certaines requêtes SQL)
- Utilisation systématique de bloc `try/except` pour les requêtes

Présentation des requêtes SQL

1) Fonction GetNB

```
def getNB(cursor, table) :  
    try :  
        cursor.execute("SELECT COUNT(0) FROM {}".format(table))  
        res = showCursor(cursor)  
        print(res)  
    except mysql.connector.Error as err :  
        print(err)
```

- *Nombre de clients total* : 99163
- *Nombre de produits total* : 32951
- *Nombre de commandes total* : 96167
- *Nombre de vendeurs* : 3088

2) Fonction getAvgNB

```
def getAvgNB(cursor) :
    try :
        cursor.execute("SELECT AVG(payment_value) FROM Order_payments")
        res = cursor.fetchall()
        print(round((res[0][0]),2))
        res = showCursor(cursor)
    except mysql.connector.Error as err :
        print(err)
```

- *Prix moyen d'une commande : 157.22*

3) Requête 'get orders by state'

```
REQUEST["get orders by state"] = """
    SELECT order_status, COUNT(*)
    FROM `Agence_KJK`.`Orders`
    GROUP BY order_status;
    """
```

- *Nombre de commandes selon leurs états :*
 - livré : 96191
 - annulé : 6

4) Requête 'get orders by month'

```
REQUEST["get orders by month"] = """
    SELECT EXTRACT(YEAR_MONTH FROM order_purchase_timestamp) AS YM, COUNT(*)
    FROM `Agence_KJK`.`Orders`
    GROUP BY YM
    ORDER BY YM
    """
```

- *Nombre de commande par mois :*

```
get orders by month
0      (201609, 1)
1      (201610, 270)
2      (201612, 1)
3      (201701, 748)
4      (201702, 1641)
5      (201703, 2540)
6      (201704, 2297)
7      (201705, 3535)
8      (201706, 3123)
9      (201707, 3865)
10     (201708, 4181)
11     (201709, 4137)
12     (201710, 4465)
13     (201711, 7267)
14     (201712, 5498)
15     (201801, 7055)
16     (201802, 6533)
17     (201803, 6981)
18     (201804, 6777)
19     (201805, 6733)
20     (201806, 6080)
21     (201807, 6139)
22     (201808, 6330)
```

5) Requête 'average orders score'

```
REQUEST["average orders score"] = ""
    SELECT AVG(review_score) FROM Order_reviews;
    ""
```

- Score de satisfaction moyen : 4.38

6) Requête 'sellers by state'

```
REQUEST["average orders score"] = ""
    SELECT AVG(review_score) FROM Order_reviews;
    ""
```

- Nombre de vendeurs par région :

```

sellers by state
0      (SP, 1847)
1      (ES, 23)
2      (RJ, 171)
3      (GO, 40)
4      (MG, 243)
5      (PR, 348)
6      (RN, 5)
7      (SC, 190)
8      (RS, 128)
9      (MA, 1)
10     (PB, 6)
11     (BA, 19)
12     (DF, 28)
13     (PE, 9)
14     (MT, 4)
15     (CE, 13)
16     (AM, 1)
17     (RO, 2)
18     (PI, 1)
19     (SE, 2)
20     (AC, 1)
21     (MS, 5)
22     (PA, 1)

```

7) Fonction newProduct

```

def newProduct(cursor, mydb) :
    try :
        cursor.execute("SELECT product_id FROM Products")
        res = showCursor(cursor)
        add = False
        while add == False:
            add = True
            newID = get_random_string(40)
            print(newID)
            for x in res :
                if x[0] == newID :
                    add = False
            print("category name :")
            newCategory = input()
            cursor.execute("""
                INSERT INTO Products VALUES (
                    '{}', '{}', {},
                    {}, {}, {},
                    {}, {}, {}
                )""".format(
                    newID, newCategory, np.random.choice(1000),
                    np.random.choice(1000), np.random.choice(1000),
                    np.random.choice(1000),
                    np.random.choice(1000), np.random.choice(1000),
                    np.random.choice(1000)
                ))
            mydb.commit()
        except mysql.connector.Error as err :
            print(err)

```

Pour créer le nouveau produit, on vérifie l'id du produit et selon si c'est True ou False, on génère l'id du produit.