

Projet de Données

ETL Analyse ROLAP

Transactions de Vente au Détail

Réalisé par :
Sankara Kabem Abdoul Charif

Outils et technologies :

- Google Colab (Python, Pandas)
- SQL Server
- Requêtes SQL (ROLAP)
- Nettoyage et transformation de données

"Un projet personnel pour explorer l'intégration de données, l'entreposage, et l'analyse décisionnelle."

5 juin 2025

Table des matières

I	Conception du modèle en étoile et construction de la base de données	3
I.1	Analyse des données	3
I.2	Définition des Dimensions	3
I.3	Construction de la Base de Données	4
II	Développement du Processus ETL	7
III	Opérations ROLAP	11
III.1	Requêtes de Type Slice	11
III.2	Requêtes de Type Dice	12
III.3	Requêtes de Type Roll-up	14
III.4	Requêtes de Type Drill-down	15
IV	Conclusion	16

Introduction

Ce rapport présente le projet P1 intitulé « Entreposage des données » réalisé dans le cadre du cours INF1473-Entreposage et prospection des données . Ce projet a pour objectif principal la conception d'un entrepôt de données basé sur un ensemble de données de transactions de détail.

L'ensemble de données, tiré du site Kaggle, offre une opportunité d'analyser les comportements d'achat des clients. En se concentrant sur l'analyse des paniers d'achat, d'explorer les modèles d'achat, d'identifier des associations entre produits, et d'optimiser les stratégies de prix et de promotions. Ce rapport décrira les différentes étapes du projet, allant de la conception d'un modèle en étoile à la mise en œuvre des opérations ROLAP, en passant par le développement d'un processus ETL.

I Conception du modèle en étoile et construction de la base de données

I.1 Analyse des données

L'analyse de cet ensemble de données vise à extraire les informations clés sur les comportements d'achat, les préférences des clients et les performances des magasins en fonction des variables fournies.

Description et Exploration des Données

- **Transaction_ID** : L'identifiant unique pour chaque transaction permet de suivre les achats individuels et de regrouper les informations en fonction des clients, des périodes et des magasins.
- **Date** : La date et l'heure d'achat sont utiles pour identifier les tendances temporelles, comme les pics de ventes saisonniers et les heures de pointe des achats.
- **Customer_Name** et **Customer_Category** : Ces informations permettent de segmenter les clients par nom et catégorie.
- **Produit** : L'ensemble des produits achetés au cours de chaque transaction permet d'analyser le panier d'achat, c'est-à-dire les combinaisons de produits achetés ensemble.
- **Total_Items** et **Total_Cost** : Ces variables donnent un aperçu des habitudes de consommation, comme le volume moyen d'achats et la valeur des transactions.
- **Payment_Method** : La méthode de paiement permet d'identifier les préférences en termes de moyens de paiement.
- **Ville** et **Store_Type** : La localisation géographique et le type de magasin permettent d'étudier les différences de comportement d'achat entre les régions et les formats de points de vente.
- **Discount_Applied** et **Promotion** : L'application des remises et des promotions est essentielle pour évaluer l'impact de ces stratégies sur les ventes et identifier les promotions les plus efficaces.
- **Saison** : La saison d'achat donne une idée des variations saisonnières dans les habitudes de consommation, permettant d'adapter les stocks et les promotions.

I.2 Définition des Dimensions

Pour la création de notre base de données, nous avons traité notre jeu de données et en avons extrait cinq tables de dimensions ainsi qu'une table de faits. Ces tables permettent une analyse complète et détaillée des ventes.

Tables de Dimensions

- **Dimension Customer** : Cette dimension permet d'analyser les ventes par client, en incluant des détails tels que le nom et la catégorie du client, la ville. Elle facilite ainsi les analyses de segmentation et de fidélisation.
- **Dimension Produit** : Permet d'analyser les ventes par produit. Elle inclut des informations spécifiques sur les produits, ce qui offre une plus grande flexibilité lors des requêtes et des analyses de tendances de consommation.
- **Dimension Temps** : La dimension temps nous permet de répondre efficacement à des requêtes sur des périodes spécifiques, comme les tendances saisonnières, mensuelles, journalières ou horaires.

- **Dimension Promotion** : Cette dimension permet d’analyser les effets des promotions et des remises sur les ventes, en fournissant des informations clés sur les types et fréquences des offres.
- **Dimension Magasin** : Permet d’obtenir des informations sur les types de magasins et leurs localisations, ce qui optimise la recherche d’informations spécifiques à chaque magasin.

Table de Faits

- **Transaction_Fait** : La table de faits des transactions contient toutes les mesures indispensables pour les analyses de ventes, ainsi que les identifiants des différentes tables de dimensions pour permettre des opérations analytiques approfondies.

Le schéma en étoile ci-dessous représente les tables et leurs relations au sein du modèle de données.

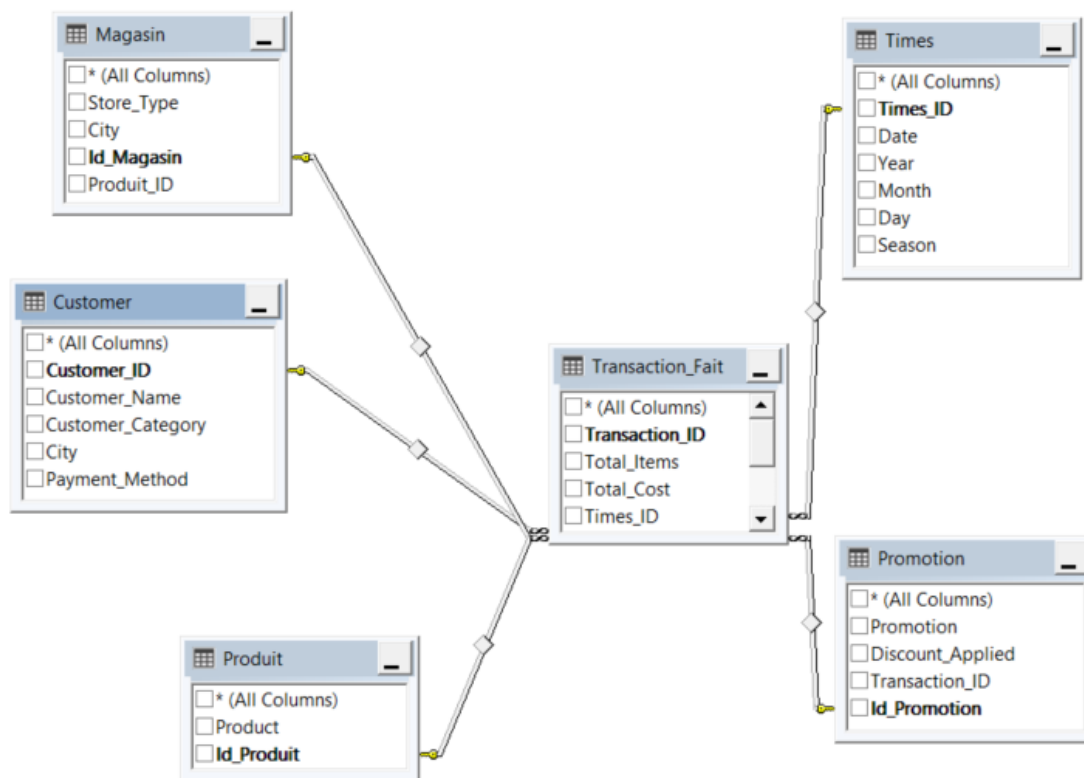


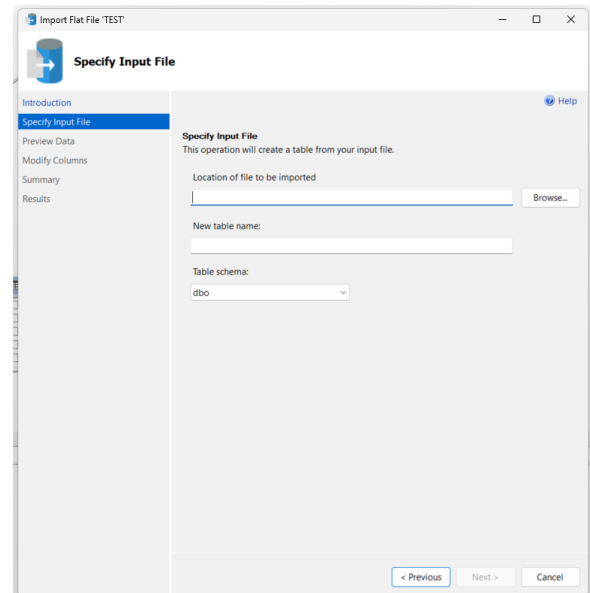
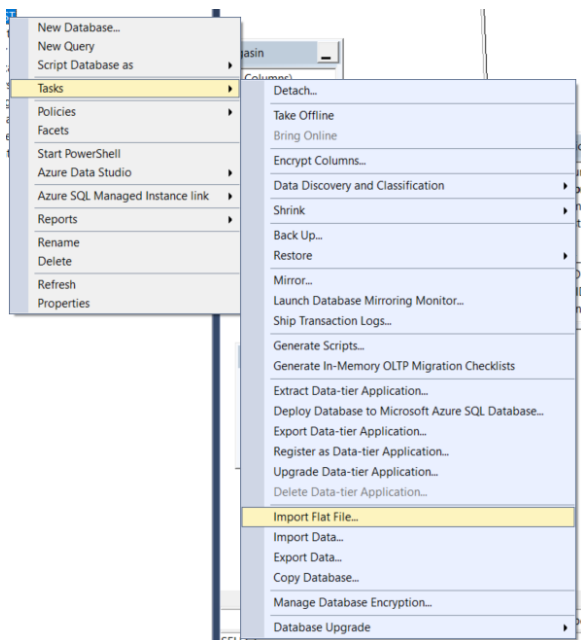
FIGURE 1 – Modèle en étoile

I.3 Construction de la Base de Données

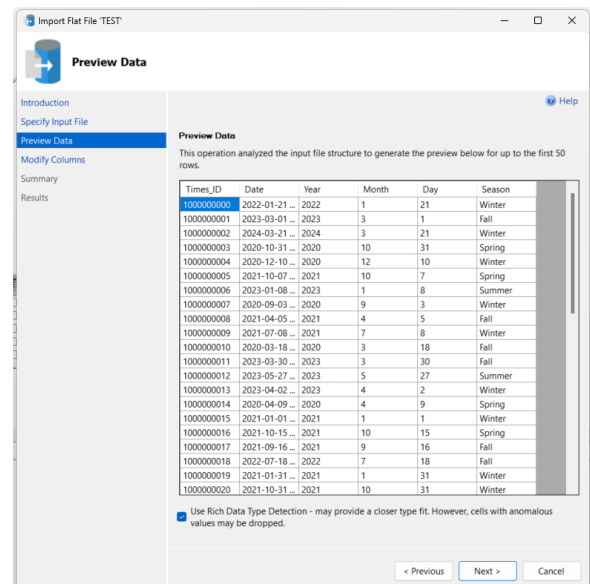
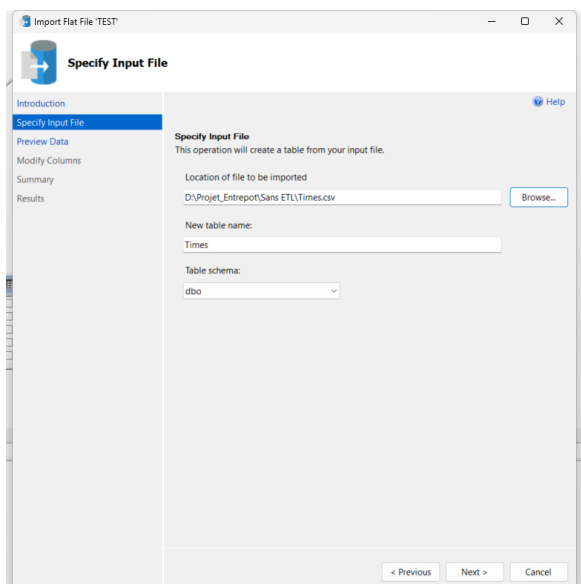
Pour construire notre base de données, une fois la conception du modèle en étoile établie, nous avons mis en œuvre un processus ETL (Extraction, Transformation, Chargement), que nous détaillerons dans le chapitre suivant. Ce processus nous a permis de transformer les données brutes pour créer nos tables de dimensions et de faits, que nous avons ensuite importées directement dans SQL Server.

Processus d'importation des données dans Sql Server

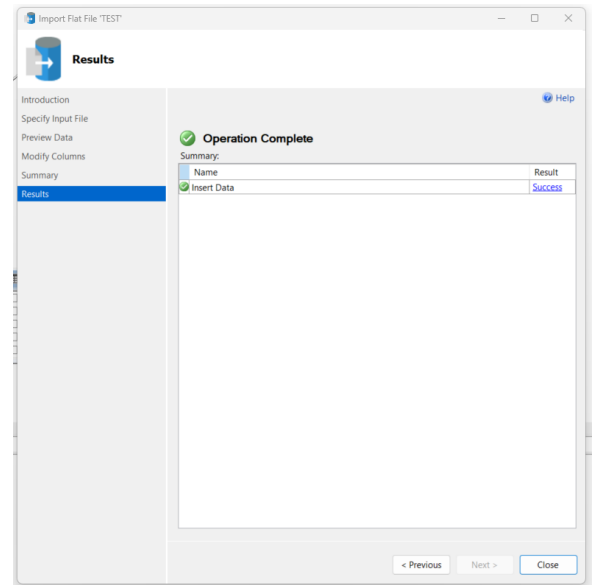
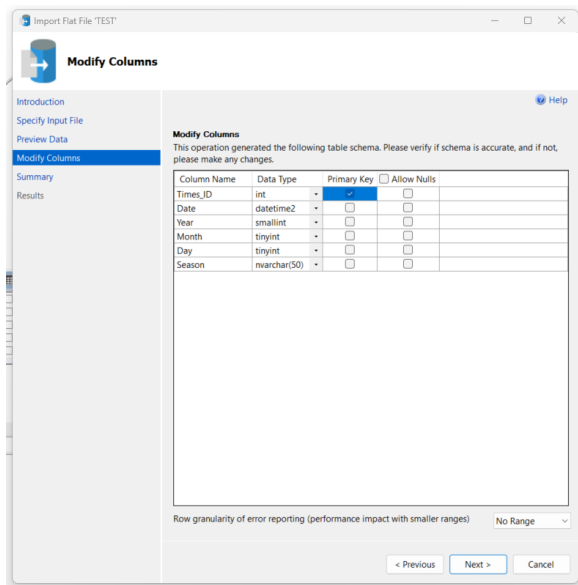
Comme expliqué précédemment, notre méthode de travail repose sur un processus ETL appliqué aux données brutes, grâce auquel nous avons obtenu les différentes tables. Les schémas ci-dessous illustrent le processus d'importation de ces tables.



Après avoir sélectionné le type de fichier, nous pouvons choisir la table de dimension souhaitée. Ici, nous avons décidé de sélectionner la table de dimension *Temps*.



Après avoir sélectionné la table et entré son nom, l'image ci-dessus montre les différentes colonnes et les données associées.



Après cela, nous pouvons choisir l'attribut qui sera désigné comme clé primaire. Nous avons également la possibilité de modifier les types de données associés.

Après avoir importé toutes les différentes tables (dimensions et faits), nous avons obtenu notre base de données, que nous avons nommée *Projet_DB*.

Le chapitre suivant présentera le processus ETL utilisé pour traiter notre jeu de données.

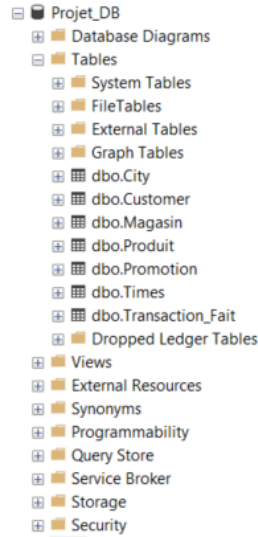


FIGURE 2 – Base de données

II Développement du Processus ETL

Pour le processus ETL de notre projet, nous avons utilisé Google Colab afin d'importer les données, de les extraire, puis de les transformer avant de les charger dans SQL Server.

Le processus ETL sera expliqué en plusieurs étapes et illustré par des captures d'écran montrant les résultats. En pièce jointe de ce document, vous trouverez le fichier .ipynb qui vous permettra de vérifier le processus. Cette section résume, en quelques points, les étapes clés du processus ETL.

Importation des données dans google colab

Après avoir importé les données à l'aide de la commande `read_csv('/content/Retail_Transaction` nous avons affiché les 20 premières lignes du jeu de données en utilisant `df.head(20)`. Cela nous a permis d'examiner la structure et le contenu du jeu de données pour mieux comprendre son organisation.

```
[1] import pandas as pd
import numpy as np

[2] from google.colab import drive
drive.mount('/content/drive')

Mounted at /content/drive
```

Chargement des données

```
df = pd.read_csv('/content/Retail_Transactions_Dataset.csv')
df.head(20)
```

	Transaction_ID	Date	Customer_Name	Product	Total_Items	Total_Cost	Payment_Method	City	Store_Type	Discount_Applied	Customer_Category	Season	Promotion
0	100000000	2022-01-21 06:27:29	Stacey Price	[Ketchup, 'Shaving Cream', 'Light Bulbs]	3	71.65	Mobile Payment	Los Angeles	Warehouse Club	True	Homemaker	Winter	NaN
1	100000001	2023-03-01 13:01:21	Michelle Carlson	['Ice Cream', 'Milk', 'Olive Oil', 'Bread', 'P...]	2	25.93	Cash	San Francisco	Specialty Store	True	Professional	Fall	BOGO (Buy One Get One)
2	100000002	2024-03-21 15:37:04	Lisa Graves	[Spinach]	6	41.49	Credit Card	Houston	Department Store	True	Professional	Winter	NaN
3	100000003	2020-10-31 09:59:47	Mrs. Patricia May	['Tissues', 'Mustard']	1	39.34	Mobile Payment	Chicago	Pharmacy	True	Homemaker	Spring	NaN
4	100000004	2020-12-10 00:59:59	Susan Mitchell	['Dish Soap']	10	16.42	Debit Card	Houston	Specialty Store	False	Young Adult	Winter	Discount on Selected Items
5	100000005	2020-10-07 12:37:26	Joshua Frazier	['Toothpaste', 'Chicken']	3	72.24	Cash	Houston	Supermarket	True	Retiree	Spring	Discount on Selected Items
6	100000006	2023-01-08 10:40:03	Victoria Garrett	['Honey', 'BBQ Sauce', 'Soda', 'Olive Oil', 'G...]	4	5.28	Cash	Boston	Specialty Store	False	Student	Summer	Discount on Selected Items
7	100000007	2020-09-03 12:39:59	Sydney Walker	['Syrup', 'Trash Cans', 'Pancake Mix', 'Water...]	5	21.77	Debit Card	Chicago	Specialty Store	False	Young Adult	Winter	Discount on Selected Items
8	100000008	2021-04-05 06:32:18	Kimberly Morgan	['Insect Repellent']	4	55.25	Mobile Payment	Los Angeles	Warehouse Club	False	Homemaker	Fall	NaN
9	100000009	2021-07-08 10:08:59	Lori Conway	['Soap', 'Baby Wipes', 'Soda']	7	31.21	Mobile Payment	Boston	Convenience Store	True	Young Adult	Winter	NaN
10	100000010	2020-03-18 18:58:18	Randall Roberts	['Extension Cords', 'Soda', 'Water', 'Garden H...]	8	91.59	Debit Card	Boston	Department Store	True	Middle-Aged	Fall	BOGO (Buy One Get One)
11	100000011	2023-03-30 15:26:41	Jeremiah White	['Tea', 'Paper Towels', 'Spinach']	7	31.66	Mobile Payment	New York	Specialty Store	False	Young Adult	Fall	Discount on Selected Items
12	100000012	2023-05-27 15:52:59	Sheila McGee	['Salmon', 'Shaving Cream']	9	59.75	Debit Card	New York	Pharmacy	False	Student	Summer	Discount on Selected Items
13	100000013	2020-04-02 09:24:11	Allen Miles	['Trash Bags', 'Apple', 'Mop', 'Hair Gel']	7	17.51	Mobile Payment	Seattle	Warehouse Club	False	Retiree	Winter	BOGO (Buy One Get One)
14	100000014	2020-04-09 04:31:05	Mark Howard	['Razors', 'Laundry Detergent', 'Beer']	1	33.64	Credit Card	San Francisco	Warehouse Club	True	Retiree	Spring	NaN
15	100000015	2021-01-01 04:29:07	Jesse Henderson	['Cereal', 'Vinegar', 'Bath Towels']	7	29.57	Mobile Payment	San Francisco	Convenience Store	False	Retiree	Winter	BOGO (Buy One Get One)
16	100000016	2021-10-15 15:24:31	Eric Wall	['Air Freshener', 'Feminine Hygiene Products']	5	99.70	Mobile Payment	Miami	Specialty Store	True	Homemaker	Spring	NaN
17	100000017	2021-09-16 09:28:28	Vanessa Peck	['Power Strips', 'Honey', 'Ketchup', 'Tea', 'S...]	8	15.62	Credit Card	San Francisco	Department Store	False	Senior Citizen	Fall	NaN
18	100000018	2022-07-18 11:33:28	Dominique Stout	['Mustard', 'Dustpan']	3	98.28	Mobile Payment	New York	Warehouse Club	False	Professional	Fall	BOGO (Buy One Get One)
19	100000019	2021-01-31 07:52:18	Charles Brooks	['Coffee']	7	96.77	Debit Card	Chicago	Department Store	True	Homemaker	Winter	Discount on Selected Items

FIGURE 3 – Importer le jeu de données

Par la suite, nous avons procédé au traitement des données en vérifiant la présence de valeurs manquantes et de données nulles. Après vérification, nous avons constaté qu'il n'y avait aucune donnée dupliquée. Cependant, nous avons identifié des données nulles dans les colonnes `Discount_Applied`, `Customer_Category` et `Season`, que nous avons traitées ultérieurement.

▼ **Traitement des données**

```

df.duplicated().sum()
df.isnull()
valeurs_nulles = df.isnull().sum()
print(valeurs_nulles)

```

```

Transaction_ID      0
Date                0
Customer_Name       0
Product             0
Total_Items         0
Total_Cost          0
Payment_Method      0
City               0
Store_Type          0
Discount_Applied    1
Customer_Category   1
Season              1
Promotion          8227
dtype: int64

```

```

[5] df.columns

```

```

Index(['Transaction_ID', 'Date', 'Customer_Name', 'Product', 'Total_Items',
      'Total_Cost', 'Payment_Method', 'City', 'Store_Type',
      'Discount_Applied', 'Customer_Category', 'Season', 'Promotion'],
      dtype='object')

```

```

[6] df['Customer_Name'].unique()

```

```

array(['Stacey Price', 'Michelle Carlson', 'Lisa Graves', ...,
      'Mr. Juan Thompson', 'Gregory Jordan', 'David Hayden'],
      dtype=object)

```

FIGURE 4 – Image 8

Dans cette section, nous avons traité la colonne `Product`. En effet, pour chaque transaction, il peut y avoir un ou plusieurs produits enregistrés. Afin d'obtenir une liste de produits, chacun dans une ligne distincte, nous avons utilisé `from itertools import chain` et appliqué la fonction `df.explode('Product')` pour séparer les produits et les placer dans des lignes individuelles.

Cependant, comme vous pouvez le constater, cela a entraîné des doublons au niveau des `Transaction_ID`, `Customer`, etc. Nous avons par la suite géré ces doublons lors de la création des tables distinctes.

▼ **Séparation des listes de produits en ligne**

```

# 1. Séparation des produits : chaque produit sera sur une ligne distincte
from itertools import chain

df['Product'] = df['Product'].str.strip('[]').str.replace("'", "").str.split(',')
df = df.explode('Product')

```

```

[9] df

```

	Transaction_ID	Date	Customer_Name	Product	Total_Items	Total_Cost	Payment_Method	City	Store_Type	Discount_Applied	Customer_Category	Season	Promotion
0	1000000000	2022-01-21 06:27:29	Stacey Price	Ketchup	3	71.65	Mobile Payment	Los Angeles	Warehouse Club	True	Homemaker	Winter	NaN
0	1000000000	2022-01-21 06:27:29	Stacey Price	Shaving Cream	3	71.65	Mobile Payment	Los Angeles	Warehouse Club	True	Homemaker	Winter	NaN
0	1000000000	2022-01-21 06:27:29	Stacey Price	Light Bulbs	3	71.65	Mobile Payment	Los Angeles	Warehouse Club	True	Homemaker	Winter	NaN
1	1000000001	2023-03-01 13:01:21	Michelle Carlson	Ice Cream	2	25.93	Cash	San Francisco	Specialty Store	True	Professional	Fall	BOGO (Buy One Get One)
1	1000000001	2023-03-01 13:01:21	Michelle Carlson	Milk	2	25.93	Cash	San Francisco	Specialty Store	True	Professional	Fall	BOGO (Buy One Get One)
...
24735	1000024735	2022-12-24 14:06:08	David Hayden	Pasta	9	6.61	Credit Card	Boston	Convenience S	NaN	NaN	NaN	NaN
24735	1000024735	2022-12-24 14:06:08	David Hayden	Hand Sanitizer	9	6.61	Credit Card	Boston	Convenience S	NaN	NaN	NaN	NaN
24735	1000024735	2022-12-24 14:06:08	David Hayden	Potatoes	9	6.61	Credit Card	Boston	Convenience S	NaN	NaN	NaN	NaN
24735	1000024735	2022-12-24 14:06:08	David Hayden	Laundry Detergent	9	6.61	Credit Card	Boston	Convenience S	NaN	NaN	NaN	NaN
24735	1000024735	2022-12-24 14:06:08	David Hayden	Mop	9	6.61	Credit Card	Boston	Convenience S	NaN	NaN	NaN	NaN

74163 rows x 13 columns

FIGURE 5 – Image 9

Après cela, nous avons transformé la colonne `Date`. Pour mieux analyser les données au

niveau temporel, nous avons utilisé la colonne Date pour créer d'autres colonnes : Year, Month, Day et Season.

▼ Traitement des dates

```

# Transformer les dates
df['Date'] = pd.to_datetime(df['Date'])

# Remplacer les valeurs NaN dans la colonne Promotion par "No Promotion"
df['Promotion'].fillna('No Promotion', inplace=True)

# Afficher les types de données après transformation
print(df.dtypes)

# 2. Transformation de la date
df['Date'] = pd.to_datetime(df['Date'])
df['Year'] = df['Date'].dt.year
df['Month'] = df['Date'].dt.month
df['Day'] = df['Date'].dt.day
df['Season'] = df['Season'].astype('category') # Saison catégorisée

df

```

	Transaction_ID	Date	Customer_Name	Product	Total_Items	Total_Cost	Payment_Method	City	Store_Type	Discount_Applied	Customer_Category	Season	Promotion	Id	Year	Month	Day
0	1000000000	2022-01-21 06:27:29	Stacey Price	Ketchup	3	71.65	Mobile Payment	Los Angeles	Warehouse Club	True	Homemaker	Winter	No Promotion	1	2022	1	21
0	1000000000	2022-01-21 06:27:29	Stacey Price	Shaving Cream	3	71.65	Mobile Payment	Los Angeles	Warehouse Club	True	Homemaker	Winter	No Promotion	2	2022	1	21
0	1000000000	2022-01-21 06:27:29	Stacey Price	Light Bulbs	3	71.65	Mobile Payment	Los Angeles	Warehouse Club	True	Homemaker	Winter	No Promotion	3	2022	1	21
1	1000000001	2023-03-01 13:01:21	Michelle Carlson	Ice Cream	2	25.93	Cash	San Francisco	Specialty Store	True	Professional	Fall	BOGO (Buy One Get One)	4	2023	3	1
1	1000000001	2023-03-01 13:01:21	Michelle Carlson	Milk	2	25.93	Cash	San Francisco	Specialty Store	True	Professional	Fall	BOGO (Buy One Get One)	5	2023	3	1
...
24735	1000024735	2022-12-24 14:06:08	David Hayden	Pasta	9	6.61	Credit Card	Boston	Convenience S	NaN	NaN	NaN	No Promotion	74159	2022	12	24
24735	1000024735	2022-12-24 14:06:08	David Hayden	Hand Sanitizer	9	6.61	Credit Card	Boston	Convenience S	NaN	NaN	NaN	No Promotion	74160	2022	12	24
24735	1000024735	2022-12-24 14:06:08	David Hayden	Potatoes	9	6.61	Credit Card	Boston	Convenience S	NaN	NaN	NaN	No Promotion	74161	2022	12	24
24735	1000024735	2022-12-24 14:06:08	David Hayden	Laundry Detergent	9	6.61	Credit Card	Boston	Convenience S	NaN	NaN	NaN	No Promotion	74162	2022	12	24
24735	1000024735	2022-12-24 14:06:08	David Hayden	Mop	9	6.61	Credit Card	Boston	Convenience S	NaN	NaN	NaN	No Promotion	74163	2022	12	24

74163 rows x 17 columns

Après avoir présenté les traitements et les transformations de certaines données, nous passons à la création de chaque table. Dans cette section, nous avons choisi de montrer la création de la table Times pour illustrer le processus de création des autres tables.

▼ Création des différentes tables

```

[16] Times = df[['Transaction_ID', 'Date', 'Year', 'Month', 'Day', 'Season']].drop_duplicates().reset_index(drop=True)

[17] Times = Times.rename(columns={'Transaction_ID': 'Times_ID'})

[18] Times

```

	Times_ID	Date	Year	Month	Day	Season
0	1000000000	2022-01-21 06:27:29	2022	1	21	Winter
1	1000000001	2023-03-01 13:01:21	2023	3	1	Fall
2	1000000002	2024-03-21 15:37:04	2024	3	21	Winter
3	1000000003	2020-10-31 09:59:47	2020	10	31	Spring
4	1000000004	2020-12-10 00:59:59	2020	12	10	Winter
...
24731	1000024731	2023-10-16 19:25:50	2023	10	16	Spring
24732	1000024732	2022-03-04 21:42:50	2022	3	4	Winter
24733	1000024733	2020-12-23 22:30:03	2020	12	23	Summer
24734	1000024734	2020-09-21 05:25:41	2020	9	21	Fall
24735	1000024735	2022-12-24 14:06:08	2022	12	24	NaN

24736 rows x 6 columns

Étapes suivantes : [Générer du code avec Times](#) [Afficher les graphiques recommandés](#) [New interactive sheet](#)

```

[19] Customer = df[['Transaction_ID', 'Customer_Name', 'Customer_Category', 'City', 'Payment_Method']].drop_duplicates().reset_index(drop=True)

[20] Customer = Customer.rename(columns={'Transaction_ID': 'Customer_ID'})

```

FIGURE 6 – Image 12

Après la création des différentes tables, nous avons procédé à l'exportation des données en vue de leur importation dans SQL Server. Nous avons déjà expliqué le processus d'importation

des tables dans SQL dans la première partie, section I.3, intitulée "Construction de la base de données".

✓ Exportation des différentes Tables

```
✓ 0 s ▶ import pandas as pd
        from google.colab import files

# Étape 2 : Enregistrement au format CSV
Transaction_fact.to_csv('Transaction_fact.csv', index=False)
Times.to_csv('Times.csv', index=False)
Magasin.to_csv('Magasin.csv', index=False)
Customer.to_csv('Customer.csv', index=False)
Produit.to_csv('Produit.csv', index=False)
Promotion.to_csv('Promotion.csv', index=False)
City.to_csv('City.csv', index=False)

# Étape 3 : Téléchargement des fichiers
files.download('Transaction_fact.csv')
files.download('Times.csv')
files.download('Magasin.csv')
files.download('Customer.csv')
files.download('Produit.csv')
files.download('Promotion.csv')
files.download('City.csv')
```

FIGURE 7 – Image 13








 Customer (1)	28/10/2024 20:53	Fichier CSV Micro...	1 345 Ko
 Promotion (1)	28/10/2024 20:53	Fichier CSV Micro...	1 041 Ko
 City (1)	28/10/2024 20:53	Fichier CSV Micro...	1 Ko
 Produit (1)	28/10/2024 20:53	Fichier CSV Micro...	3 Ko
 Times (1)	28/10/2024 20:53	Fichier CSV Micro...	1 146 Ko
 Transaction_fact (2)	28/10/2024 20:53	Fichier CSV Micro...	1 103 Ko
 Magasin (1)	28/10/2024 20:52	Fichier CSV Micro...	2 Ko

FIGURE 8 – Image 14

III Opérations ROLAP

Après le processus ETL et le chargement des données dans SQL Server, nous procédons à effectuer des opérations de Slice, Dice, roll-up et drill-down pour extraire des informations pertinentes sur les données stockées dans notre entrepôt de données.

III.1 Requêtes de Type Slice

1) Les transactions effectuées par les clients dans la ville de HOUSTON.

```
SELECT tf.Transaction_ID, tf.Total_Cost, c.Customer_Name
FROM Transaction_Fait tf
JOIN Customer c ON tf.Customer_ID = c.Customer_ID
WHERE c.City = 'Houston';
```

	Transaction_ID	Total_Cost	Customer_Name
1	1000106025	23	Maria Flores
2	1000106027	91	Charlene Munoz
3	1000106038	14	Jeff Garcia
4	1000106052	86	Diane Moreno
5	1000106058	72	Garrett Singh
6	1000106089	79	Jennifer Hernandez
7	1000106090	56	Brenda Sawyer
8	1000106096	67	Jennifer Webster
9	1000106128	30	Stephanie Reed
10	1000106131	74	Bruce Kelley
11	1000106133	20	Gina Carrillo
12	1000106137	68	Robert Rivera
13	1000106138	64	Raven Miller
14	1000106144	97	William Welch
15	1000106148	50	Heather Bush
16	1000106161	13	Carol Stanley
17	1000106168	93	Danny Moore
18	1000106176	35	Mary Banks
19	1000106180	26	Teresa Johnson
20	1000106187	48	Robert Moore

Transactions :

Les montants de transaction varient considérablement, allant de 13 à 97. Cela indique une diversité dans les achats, ce qui pourrait refléter des comportements d'achat différents parmi les clients.

Clients avec les Montants les Plus Élevés :

Les clients avec les transactions les plus élevées incluent :

- **William Welch** avec un montant de 97
- **Diane Moreno** avec un montant de 86

Ces clients pourraient représenter une cible importante pour les promotions ou les programmes de fidélité, car ils semblent avoir un potentiel d'achat élevé.

Clients avec les Montants les Plus Bas :

À l'inverse, **Carol Stanley** a effectué une transaction d'un montant de 13, ce qui pourrait indiquer un comportement d'achat moins engagé ou un achat ponctuel. Cela pourrait nécessiter une stratégie de ré-engagement.

2) Afficher les transactions avec un coût total supérieur à 10

```
SELECT tf.Transaction_ID, tf.Total_Cost, tf.Total_Items
FROM Transaction_Fait tf
WHERE tf.Total_Cost > 10;
```

Results Messages			
	Transaction_ID	Total_Cost	Total_Items
1	1000000000	71	3
2	1000000001	25	2
3	1000000002	41	6
4	1000000003	39	1
5	1000000004	16	10
6	1000000005	72	3
7	1000000007	21	5
8	1000000008	55	4
9	1000000009	31	7
10	1000000010	91	8
11	1000000011	31	7
12	1000000012	39	9
13	1000000013	17	7
14	1000000014	33	1
15	1000000015	29	7
16	1000000016	99	5
17	1000000017	15	8
18	1000000018	98	3
19	1000000019	96	7
20	1000000020	83	7

3) Transactions pendant l'été

```
SELECT *
FROM Transaction_Fait tf
JOIN Times t ON tf.Times_ID = t.Times_ID
WHERE t.Season = 'Summer';
```

Results Messages														
	Transaction_ID	Total_Items	Total_Cost	Times_ID	Customer_ID	Product_ID	Magasin_ID	Promotion_ID	Times_ID	Date	Year	Month	Day	Season
1	1000052994	10	64	1000052994	1000052994	NULL	NULL	52995	1000052994	2021-05-09 19:43:17.00000000	2021	5	9	Summer
2	1000052997	1	98	1000052997	1000052997	NULL	NULL	52998	1000052997	2023-04-28 12:13:50.00000000	2023	4	28	Summer
3	1000053004	1	19	1000053004	1000053004	NULL	NULL	53005	1000053004	2023-02-03 13:05:46.00000000	2023	2	3	Summer
4	1000053010	5	5	1000053010	1000053010	NULL	NULL	53011	1000053010	2023-04-17 16:11:33.00000000	2023	4	17	Summer
5	1000053023	1	65	1000053023	1000053023	NULL	NULL	53024	1000053023	2023-06-27 04:47:16.00000000	2023	6	27	Summer
6	1000053024	9	50	1000053024	1000053024	NULL	NULL	53025	1000053024	2021-07-16 00:44:08.00000000	2021	7	16	Summer
7	1000053026	1	99	1000053026	1000053026	NULL	NULL	53027	1000053026	2021-05-18 19:06:21.00000000	2021	5	18	Summer
8	1000053027	5	70	1000053027	1000053027	NULL	NULL	53028	1000053027	2021-06-01 08:23:13.00000000	2021	6	1	Summer
9	1000053031	1	34	1000053031	1000053031	NULL	NULL	53032	1000053031	2023-07-18 03:45:07.00000000	2023	7	18	Summer
10	1000053033	4	98	1000053033	1000053033	NULL	NULL	53034	1000053033	2022-11-09 00:15:28.00000000	2022	11	9	Summer
11	1000053034	9	47	1000053034	1000053034	NULL	NULL	53035	1000053034	2023-06-05 14:41:14.00000000	2023	6	5	Summer
12	1000053035	9	11	1000053035	1000053035	NULL	NULL	53036	1000053035	2020-05-19 06:57:36.00000000	2020	5	19	Summer
13	1000053042	7	61	1000053042	1000053042	NULL	NULL	53043	1000053042	2023-10-17 09:12:49.00000000	2023	10	17	Summer
14	1000053046	5	19	1000053046	1000053046	NULL	NULL	53047	1000053046	2022-02-23 03:53:14.00000000	2022	2	23	Summer
15	1000053050	10	53	1000053050	1000053050	NULL	NULL	53051	1000053050	2021-06-05 09:12:04.00000000	2021	6	5	Summer
16	1000053051	6	10	1000053051	1000053051	NULL	NULL	53052	1000053051	2023-07-14 05:06:54.00000000	2023	7	14	Summer
17	1000053054	10	86	1000053054	1000053054	NULL	NULL	53055	1000053054	2022-12-01 02:22:36.00000000	2022	12	1	Summer
18	1000053056	5	62	1000053056	1000053056	NULL	NULL	53057	1000053056	2020-05-28 22:31:05.00000000	2020	5	28	Summer
19	1000053058	8	22	1000053058	1000053058	NULL	NULL	53059	1000053058	2023-02-18 05:47:28.00000000	2023	2	18	Summer
20	1000053060	5	97	1000053060	1000053060	NULL	NULL	53061	1000053060	2022-05-24 05:26:15.00000000	2022	5	24	Summer

III.2 Requêtes de Type Dice

1) Transactions par des clients de catégorie "Homemaker" avec paiement par "Cash"

```
SELECT tf.Transaction_ID, tf.Total_Cost, c.Customer_Name
FROM Transaction_Fait tf
JOIN Customer c ON tf.Customer_ID = c.Customer_ID
WHERE c.Customer_Category = 'Homemaker' AND c.Payment_Method = 'Cash';
```

	Transaction_ID	Total_Cost	Customer_Name
1	1000053008	91	Curtis Walker
2	1000053017	60	Elizabeth Cox
3	1000053045	69	Jordan Robinson
4	1000053070	13	Thomas Gray
5	1000053113	92	Cynthia Rose
6	1000053206	96	Joseph Reyes
7	1000053213	49	Teresa Sanders
8	1000053267	25	Jennifer Hamilton
9	1000053349	66	Paula Manning
10	1000053376	29	Jesus Berry
11	1000053383	33	Peggy Silva
12	1000053386	68	Jaime Mitchell
13	1000053453	98	Robin Martinez DDS
14	1000053465	83	Nicholas Rodriguez
15	1000053492	67	Maureen Kim
16	1000053498	99	Rick Parsons
17	1000053502	19	Katherine Lewis
18	1000053571	54	Erin Thomas
19	1000053627	52	John Carr
20	1000053636	77	Jacob Gonzalez

2) Transactions de clients de catégorie "Professionnel" avec remise

```
SELECT tf.Transaction_ID, tf.Total_Items, tf.Total_Cost, c.Customer_Category, p.
FROM Transaction_Fait tf
JOIN Customer c ON tf.Customer_ID = c.Customer_ID
JOIN Promotion p ON tf.Promotion_ID = p.Id_Promotion
WHERE c.Customer_Category = 'Professional' AND p.Discount_Applied = 1;
```

	Transaction_ID	Total_Items	Total_Cost	Customer_Category	Promotion
1	1000314944	6	80	Professional	Discount on Selected Items
2	1000314954	2	82	Professional	Discount on Selected Items
3	1000314955	7	94	Professional	BOGO (Buy One Get One)
4	1000315019	9	12	Professional	Discount on Selected Items
5	1000315037	10	66	Professional	No Promotion
6	1000315051	4	18	Professional	Discount on Selected Items
7	1000315079	1	68	Professional	BOGO (Buy One Get One)
8	1000315121	5	55	Professional	BOGO (Buy One Get One)
9	1000315136	10	33	Professional	Discount on Selected Items
10	1000315140	10	91	Professional	Discount on Selected Items
11	1000315141	10	81	Professional	No Promotion
12	1000315170	1	50	Professional	BOGO (Buy One Get One)
13	1000315185	7	95	Professional	BOGO (Buy One Get One)
14	1000315192	6	87	Professional	BOGO (Buy One Get One)
15	1000315202	1	27	Professional	BOGO (Buy One Get One)
16	1000315243	1	82	Professional	Discount on Selected Items
17	1000315245	10	76	Professional	BOGO (Buy One Get One)
18	1000315246	2	42	Professional	No Promotion
19	1000315283	6	46	Professional	BOGO (Buy One Get One)
20	1000315304	2	97	Professional	No Promotion

III.3 Requêtes de Type Roll-up

1) Total des ventes par année

```
SELECT t.Year, SUM(tf.Total_Cost) AS Total_Sales
FROM Transaction_Fait tf
JOIN Times t ON tf.Times_ID = t.Times_ID
GROUP BY t.Year
ORDER BY t.Year;
```

	Year	Total_Sales
1	2020	11902385
2	2021	11855837
3	2022	11834247
4	2023	11870327
5	2024	4497326

2) Ventes par type de magasin

```
SELECT m.Store_Type, SUM(tf.Total_Cost) AS Total_Sales
FROM Transaction_Fait tf
JOIN Magasin m ON tf.Magasin_ID = m.Id_Magasin
GROUP BY m.Store_Type
ORDER BY Total_Sales DESC;
```

	Store_Type	Total_Sales
1	Department Store	690
2	Pharmacy	489
3	Supermarket	434
4	Convenience Store	382
5	Specialty Store	332
6	Warehouse Club	316

3) Ventes totales par mois

```
SELECT t.Month, SUM(tf.Total_Cost) AS Total_Sales
FROM Transaction_Fait tf
JOIN Times t ON tf.Times_ID = t.Times_ID
GROUP BY t.Month
ORDER BY t.Month;
```

	Month	Total_Sales
1	1	5036110
2	2	4603981
3	3	5058937
4	4	4863710
5	5	4591509
6	6	3905576
7	7	4041907
8	8	4016541
9	9	3885437
10	10	4021538
11	11	3926306
12	12	4008570

III.4 Requetes de Type Drill-down

1) Détails des ventes pour un magasin spécifique (Convenience Store) en 2023

```
SELECT t.Date, tf.Total_Cost, tf.Total_Items
FROM Transaction_Fait tf
JOIN Times t ON tf.Times_ID = t.Times_ID
JOIN Magasin m ON tf.Magasin_ID = m.Id_Magasin
WHERE m.Store_Type = 'Convenience_Store' AND t.Year = 2023
ORDER BY t.Date;
```

Results		Messages	
	Date	Total_Cost	Total_Items
1	2023-02-27 16:46:56.00000000	24	4
2	2023-04-04 10:57:09.00000000	41	7

2) Analyse des ventes journalières dans un mois de Juin

```
SELECT t.Date, SUM(tf.Total_Cost) AS Daily_Sales
FROM Transaction_Fait tf
JOIN Times t ON tf.Times_ID = t.Times_ID
WHERE t.Month = '07'
GROUP BY t.Date
ORDER BY t.Date;
```


Results		Messages
	Date	Daily_Sales
1	2020-07-01 00:01:04.0000000	64
2	2020-07-01 00:01:47.0000000	29
3	2020-07-01 00:02:02.0000000	44
4	2020-07-01 00:04:09.0000000	18
5	2020-07-01 00:06:05.0000000	81
6	2020-07-01 00:06:23.0000000	58
7	2020-07-01 00:07:47.0000000	17
8	2020-07-01 00:09:10.0000000	40
9	2020-07-01 00:15:23.0000000	56
10	2020-07-01 00:19:24.0000000	70
11	2020-07-01 00:19:36.0000000	6
12	2020-07-01 00:29:26.0000000	23
13	2020-07-01 00:29:36.0000000	74
14	2020-07-01 00:29:50.0000000	20
15	2020-07-01 00:32:35.0000000	27
16	2020-07-01 00:37:15.0000000	90
17	2020-07-01 00:38:53.0000000	10
18	2020-07-01 00:39:19.0000000	91
19	2020-07-01 00:41:21.0000000	23
20	2020-07-01 00:42:01.0000000	23

✓ Query executed successfully.

IV Conclusion

Ce projet a permis de développer un système de gestion de base de données efficace pour analyser les transactions de vente au détail. À travers un processus ETL bien structuré, nous avons extrait, transformé et chargé des données provenant de diverses sources, garantissant leur intégrité et leur pertinence pour les analyses futures.

Nous avons créé plusieurs tables, notamment des tables de dimensions (Times, Customer, Product, Promotion, Magasin) et une table de faits (Transaction_Fait), qui nous permettent de réaliser des analyses approfondies. Les opérations de Slice, Dice, Roll-up et Drill-down ont été mises en œuvre pour extraire des informations significatives, permettant ainsi de mieux comprendre les tendances de vente et le comportement des clients.

L'utilisation d'outils tels que Google Colab a facilité le traitement des données et leur exportation vers SQL Server. Grâce à une visualisation claire des résultats, nous avons pu identifier des modèles de consommation, évaluer l'impact des promotions et optimiser les stratégies de vente.

En somme, ce projet constitue une étape significative vers une prise de décision basée sur les données, offrant des perspectives précieuses pour l'amélioration continue des performances commerciales et le développement de stratégies marketing ciblées.