République Tunisienne
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université de Sfax

Faculté des Sciences de Sfax
Département d'Informatique et de Communications

Projet de Mise en Œuvre d'un Entrepôt de Données

L'analyse des CONSULTATIONS

Préparé par:
Mohamed Amine Salhi
&
Oussema Zghal

Étudiants en 1ère année du cycle ingénieur en informatique.

Encadré par:

Mme. Senda Bouaziz

Ms. Ali Salem

Academic Year:

2024/2025

Table des Matières

5. **Étape 5 : Analyse des Données avec Power BI et Excel**
   5.1 Introduction
   5.2 Analyse avec Power BI
   5.3 Analyse avec Excel
   5.4 Exploration des Données à l'Aide de Visualisations
   5.5 Résultats de l'Analyse et Interprétation des Données

6. **Conclusion Finale**
   6.1 Récapitulatif des Étapes Clés
   6.2 Impact et Résultats

Étape 0 : Conception de l'entrepôt de données

La conception est une étape essentielle pour structurer les données et garantir un accès efficace aux informations nécessaires à l'analyse. Dans cette phase, nous avons défini les objectifs, identifié les sources de données, et modélisé la structure logique de l'entrepôt sous forme de schéma conceptuel.

Analyse des besoins

L'objectif principal de cet entrepôt de données est de centraliser et d'organiser les informations provenant de diverses sources (Access, JSON, CSV) pour permettre une analyse facile et rapide. Les besoins identifiés incluent :

- **Consolidation des données** : Intégrer des informations issues de différents fichiers pour obtenir une vue d'ensemble homogène.
- **Facilité d'analyse** : Organiser les données en dimensions claires pour simplifier les analyses futures.
- **Support pour les analyses décisionnelles** : Fournir des données prêtes pour les tableaux de bord et les rapports.

Identification des sources de données

Nous avons identifié trois principales sources de données brutes :

1. **Access** : Contient des informations sur les consultations, comme les patients, les médecins, et les hôpitaux (année 2024).
2. **JSON** : Contient des détails géographiques tels que les villes et pays (année 2023).
3. **CSV** : Contient des données supplémentaires sur les patients et les médecins. (année 2020-2022).

Ces fichiers sont hétérogènes en termes de structure et de format, ce qui rend l'intégration cruciale.

Modélisation des dimensions et de la table des faits

Nous avons utilisé un **modèle en étoile** pour la conception de l'entrepôt de données. Ce modèle simplifie les analyses en organisant les données autour d'une table centrale (table des faits) liée à plusieurs tables de dimensions.

- **Table des faits :**
    - Cette table contient les mesures clés, comme le prix et la durée des consultations.

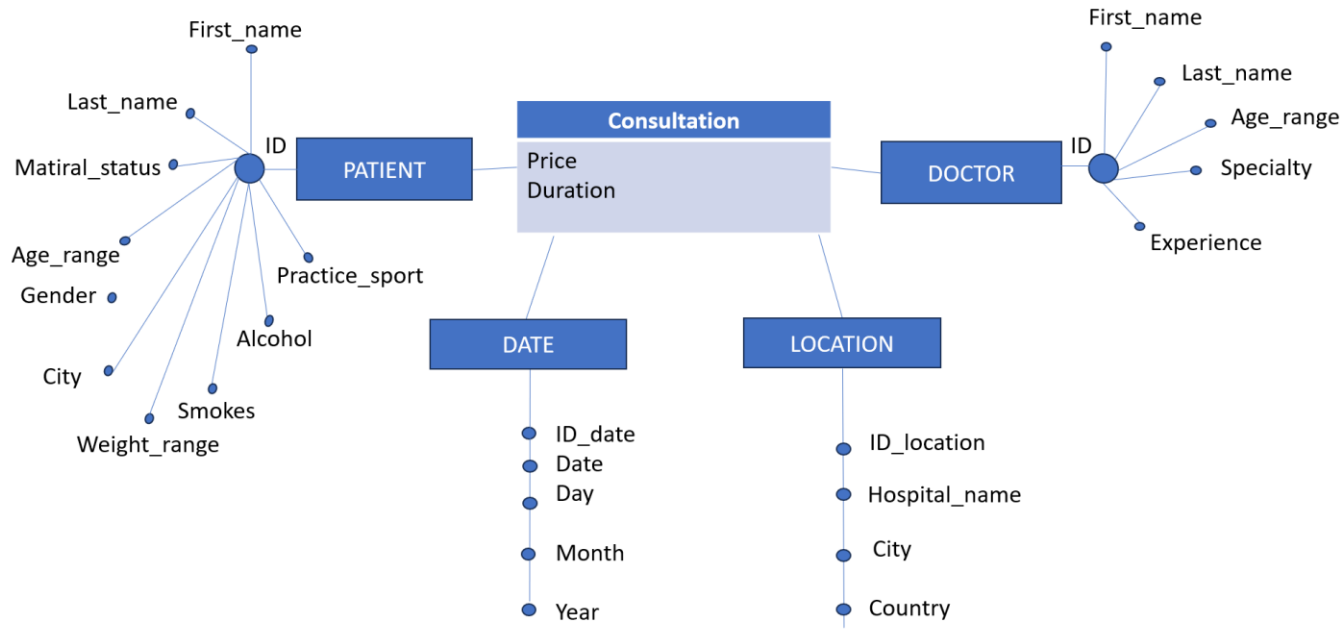
- Colonnes principales : ID_patient, ID_doctor, ID_location, ID_time, Price, Duration.

- **Dimensions principales :**

1. **Dimension Patient** :
    - Données : Nom complet, tranche d'âge, poids, état de santé.
2. **Dimension Médecin** :
    - Données : Nom complet, âge, spécialité.
3. **Dimension Location** :
    - Données : Hôpital, ville, pays.
4. **Dimension Temps** :
    - Données : Année, mois, jour de la semaine.

Schéma conceptuel

Le schéma ci-dessous, conçu avec PowerPoint, illustre la structure de l'entrepôt de données en suivant le modèle en étoile. Il montre les relations entre la table des faits et les dimensions.



Étape 1 : Filtrage et Extraction des Données

1.1 Introduction

Lors de cette étape, nous avons appliqué des transformations spécifiques sur les fichiers **Access**, **JSON**, et **CSV** afin de standardiser les données et faciliter leur intégration dans un entrepôt de données. Deux méthodes ont été utilisées pour réaliser cette étape :

1. Avec **Talend Open Studio**.
2. Avec un **code Python** personnalisé.

1.2 Détails des Transformations

1.2.1 Fichier Access

Pour le fichier Access, les colonnes **practice_sport**, **smokes**, et **alcohol** contenaient des valeurs sous forme de chaînes de caractères ("1" ou autres). Ces valeurs ont été transformées en booléens :

- "1" a été remplacé par **True**.
- Toutes les autres valeurs ont été remplacées par **False**.

1.2.2 Fichier JSON

Pour le fichier JSON, les colonnes **city** et **country** ont été formatées de manière spécifique :

- **Initialement**: La fonction était limitée à afficher uniquement les trois premiers caractères de chaque valeur dans les colonnes **city** et **country**.
- **Modification :** Nous avons ajusté la fonction pour qu'elle affiche la valeur complète dans ces colonnes. Ainsi, toutes les données sont désormais correctement affichées sans aucune limitation de taille.

1.2.3 Fichier CSV

Pour le fichier CSV, des transformations similaires à celles effectuées sur le fichier Access ont été appliquées :

- Les colonnes **practice_sport**, **smokes**, et **alcohol**, initialement sous forme de chaînes de caractères, ont été converties en booléens (comme dans le fichier Access).

1.3 Méthodes Utilisées

1.3.1 Méthode avec Talend Open Studio

Avec Talend, nous avons procédé comme suit :

1. **Filtrage Spécifique :**
    - À l'aide du composant **tMap**, nous avons appliqué des règles spécifiques de filtrage pour chaque type de ressource (Access, JSON, CSV) selon les transformations décrites ci-dessus.
    - Une capture d'écran du tMap utilisé est donnée ci-dessous pour illustrer cette étape.
2. **Jointure des Données :**
    - Le composant **tUnite** a été utilisé pour combiner les données des trois sources dans un fichier CSV unifié.

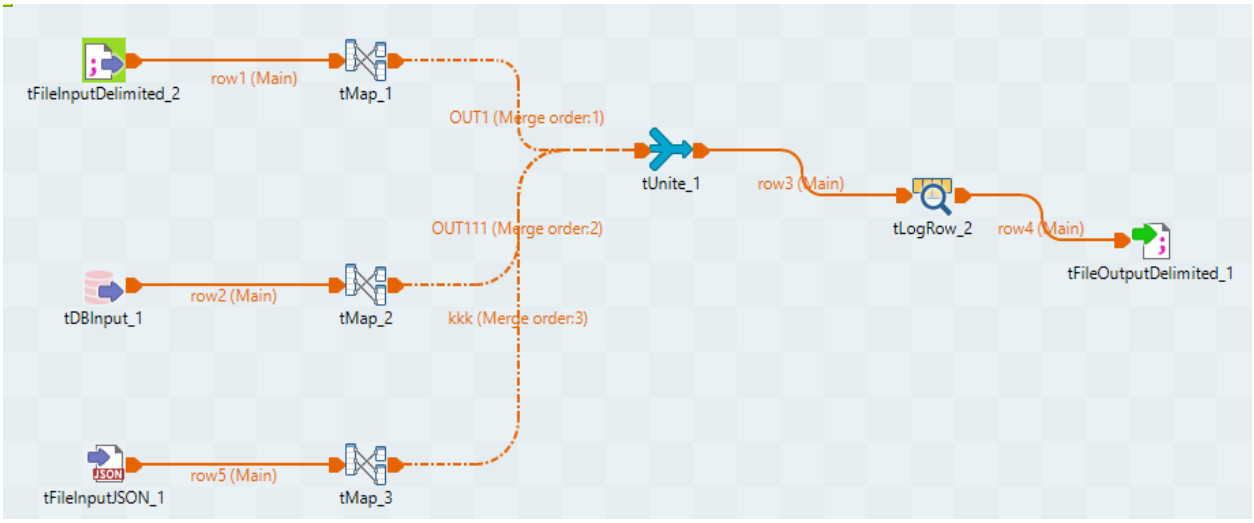
Le fichier final obtenu avec Talend contenait toutes les données filtrées et transformées.

1.3.2 Méthode avec Python

En complément de Talend, nous avons également développé un script Python qui réalise les mêmes étapes :

1. **Filtrage :**
    - Pour chaque fichier (Access, JSON, CSV), nous avons écrit des fonctions spécifiques pour effectuer les transformations nécessaires et sauvegarder une copie des fichiers filtrés.
2. **Jointure :**
    - Un second script a été développé pour combiner les fichiers filtrés dans un fichier CSV final.

1.4 Capture d'écran : Méthode avec Talend



1.5 Code Python Expliqué

Voici un extrait de code Python utilisé pour effectuer le filtrage et la jointure des fichiers :

Filtrage des fichiers Access, JSON, et CSV

Json:

```python
import pandas as pd
import json

# Remplacer avec le nom complet si la valeur existe
def replace_with_full_name(value):
    return f"Full Name: {value}" if value else value

# Charger les données JSON
def load_json_data(file):
    with open(file, 'r') as f:
        return pd.json_normalize(json.load(f))

# Appliquer le filtrage sur 'city' et 'country'
def filter_data(df):
    for col in ['city', 'country']:
        df[col] = df[col].apply(replace_with_full_name)
    return df

# Exporter les données filtrées au format JSON
def export_data(df, output_file):
    df.to_json(output_file, orient='records', lines=True)
    print(f"Données exportées vers {output_file}.")

#Execution
input_file = "C:/Users/Med Amine Salhi/Desktop/datawarehouse_salhi/datawarehouse_salhi/fichier_filtre_2023_modifie.json"
output_file = input_file  # Même fichier de sortie

# Charger et filtrer les données, puis exporter
data = load_json_data(input_file)
filtered_data = filter_data(data)
export_data(filtered_data, output_file)
```

Access:

```python
import pandas as pd
import pyodbc

# Fonction pour charger les données depuis Access
def load_access_data(access_file, table_name):
    connection_string = f"DRIVER={{Microsoft Access Driver (*.mdb, *.accdb)}};DBQ={access_file};"
    conn = pyodbc.connect(connection_string)
    query = f"SELECT * FROM {table_name}"
    data = pd.read_sql(query, conn)
    conn.close()
    return data

# Fonction de filtrage pour les colonnes practice_sport, smokes, alcohol
def filter_data(df):
    # Remplacer les valeurs "1" par True et les autres par False
    df['practice_sport'] = df['practice_sport'].apply(lambda x: True if x == '1' else False)
    df['smokes'] = df['smokes'].apply(lambda x: True if x == '1' else False)
    df['alcohol'] = df['alcohol'].apply(lambda x: True if x == '1' else False)
    return df

# Exporter les données filtrées dans un fichier CSV
def export_to_csv(df, output_file):
    df.to_csv(output_file, index=False)
    print(f"Les données filtrées ont été exportées vers {output_file}.")

#Execution
access_file = "C:/Users/Med Amine Salhi/Desktop/datawarehouse_salhi/datawarehouse_salhi/Database91.accdb"
table_name = "Consultations"
output_file = "C:/Users/Med Amine Salhi/Desktop/datawarehouse_salhi/filtered_output.csv"  # Fichier de sortie

# Charger les données depuis Access
access_data = load_access_data(access_file, table_name)

# Appliquer le filtrage sur les colonnes practice_sport, smokes, alcohol
filtered_data = filter_data(access_data)

# Exporter les données filtrées
export_to_csv(filtered_data, output_file)
```

Jointure des fichiers filtrés

```python
import pandas as pd
import pyodbc
import json

# Charger des données depuis une base Access
def load_access_data(file, table):
    conn = pyodbc.connect(f"DRIVER={{Microsoft Access Driver (*.mdb, *.accdb)}};DBQ={file};")
    data = pd.read_sql(f"SELECT * FROM {table}", conn)
    conn.close()
    return data

# Charger des données depuis un fichier JSON
def load_json_data(file):
    with open(file, 'r') as f:
        return pd.json_normalize(json.load(f))

# Charger des données depuis un fichier CSV
def load_csv_data(file):
    return pd.read_csv(file)

# Fusionner trois DataFrames sur une clé commune
def merge_data(df1, df2, df3, key):
    return pd.merge(pd.merge(df1, df2, on=key), df3, on=key)

# Exporter un DataFrame vers un fichier CSV
def export_to_csv(df, file):
    df.to_csv(file, index=False)
    print(f"Données exportées : {file}")

# Script principal
def main():
    # Chemins des fichiers et paramètres
    access_file = "C:/Users/Med Amine Salhi/Desktop/datawarehouse_salhi/Database91.accdb"
    json_file = "C:/Users/Med Amine Salhi/Desktop/datawarehouse_salhi/fichier_filtre_2023_modifie.json"
    csv_file = "C:/Users/Med Amine Salhi/Desktop/datawarehouse_salhi/fichier_sans_2023.csv"
    output_file = "C:/Users/Med Amine Salhi/Desktop/projet entrepot/output.csv"
    table_name = "Consultations"
    key_column = "id"

    # Chargement des données
    access_data = load_access_data(access_file, table_name)
    json_data = load_json_data(json_file)
    csv_data = load_csv_data(csv_file)

    # Vérification des données chargées
    print(access_data.head(), json_data.head(), csv_data.head(), sep="\n\n")

    # Fusion des données
    merged_data = merge_data(access_data, json_data, csv_data, key_column)

    # Affichage des données fusionnées
    print(merged_data.head())

    # Exportation des données fusionnées
    export_to_csv(merged_data, output_file)
# Appel de la fonction principale
main()
```

1.6 Résultat

Le fichier final généré contient toutes les données des trois sources (Access, JSON, CSV) après transformation et jointure, prêtes à être utilisées dans l'entrepôt de données.

2ème Étape : Extraction des Dimensions et Chargement dans la Base de Données (MySQL)

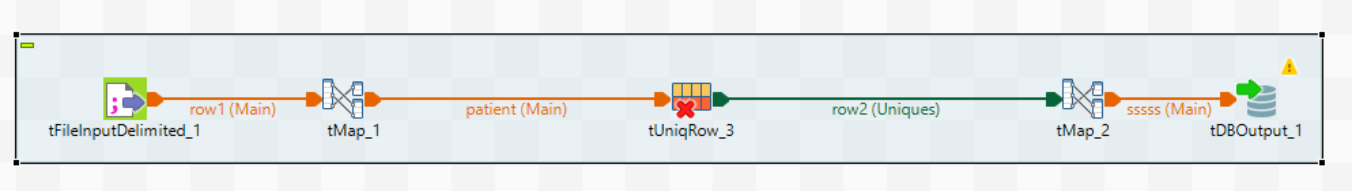
2.1 Introduction

Dans cette étape, nous avons extrait quatre dimensions principales à partir du **DATASET** créé dans la première étape et les avons chargées dans la base de données **MySQL**. Les dimensions extraites sont :

1. **Patient**
2. **Médecin**
3. **Location**
4. **Date (Temps)**

Pour chaque dimension, nous avons effectué des transformations nécessaires pour garantir la qualité des données et respecter la granularité de chaque dimension.

2.2 La Dimension Patient



2.2.1 Transformation des Colonnes

Pour la dimension **Patient**, plusieurs transformations ont été appliquées :

1. **Création de la colonne person_full_name :** Nous avons combiné les colonnes **person_first_name** et **person_last_name** en une seule colonne **person_full_name**. Cette transformation a été réalisée de deux manières :
    - **Avec Talend** : La fonction utilisée dans Talend est :

      row1.person_first_name + " " + row1.person_last_name

    - **Avec Python** : Pour la partie Python, voici la transformation :

```
df['person_full_name'] = df['person_first_name'] + ' ' + df['person_last_name']
```

2. **Transformation du Poids (weight)** : Nous avons normalisé le **poids** en utilisant la fonction **getWeightRange** :

      routine1.getWeightRange((row1.weight) / 2)

Cela permet de classer les poids en fonction de critères définis dans la fonction getWeightRange.

3. **Transformation de l'Âge (person_age)** : Nous avons également transformé l'âge du patient en utilisant la fonction **getAgeRange** :

routine1.getAgeRange(row1.person_age)

Cette transformation permet de classer les patients par tranche d'âge.

4. **Ajout de la Colonne id** : Une colonne **id** a été ajoutée à la dimension **Patient**. Cette colonne est générée automatiquement et incrémentée de 1 pour chaque nouvelle ligne, afin de faciliter la manipulation et l'intégration dans la base de données.

2.2.2 Charge dans MySQL

Après les transformations, les données sont chargées dans la table **Patient** de la base de données MySQL. Chaque enregistrement contient l'identifiant **id**, le nom complet **person_full_name**, le poids transformé, l'âge transformé, et d'autres informations pertinentes.

2.2.3 Code python

1. **Extraction de la dimension**

```python
import pandas as pd

# Charger le fichier source total_data.csv
df_total = pd.read_csv('C:/Users/Med Amine Salhi/Desktop/datawarehouse_salhi/datawarehouse_salhi/tous.csv')

#Extraction des colonnes:
df_patient = df_total[['person_first_name', 'person_last_name', 'person_age', 'job', 'marital_status', 'gender', 'weight', 'practice_sport', 'smokes', 'alcohol']]

# Sauvegarder les données extraites dans un fichier CSV dédié à la dimension Patient
df_patient.to_csv('C:/Users/Med Amine Salhi/Desktop/datawarehouse_salhi/datawarehouse_salhi/patient_data.csv', index=False)
```

2. **Transformation**

```python
# Charger le fichier CSV
df_patient = pd.read_csv('C:/Users/Med Amine Salhi/Desktop/datawarehouse_salhi/datawarehouse_salhi/patient_data.csv')

# Assurez-vous que les colonnes 'weight' et 'person_age' sont numériques
df_patient['weight'] = pd.to_numeric(df_patient['weight'], errors='coerce')
df_patient['person_age'] = pd.to_numeric(df_patient['person_age'], errors='coerce')

# 1. Création de la colonne `person_full_name`
df_patient['person_full_name'] = df_patient['person_first_name'] + ' ' + df_patient['person_last_name']

# 2. Transformation du poids (`weight`)
def get_weight_range(weight):
    if weight < 50:
        return 'Underweight'
    elif weight < 100:
        return 'Normal weight'
    elif weight < 150:
        return 'Overweight'
    else:
        return 'Obese'

df_patient['weight_range'] = df_patient['weight'].apply(get_weight_range)

# 3. Transformation de l'âge (`person_age`)
def get_age_range(age):
    if age < 18:
        return 'Minor'
    elif age < 30:
        return 'Young Adult'
    elif age < 60:
        return 'Adult'
    else:
        return 'Senior'

df_patient['age_range'] = df_patient['person_age'].apply(get_age_range)

# 4. Ajout de la colonne `id`
df_patient['id'] = range(1, len(df_patient) + 1)

# Afficher les données transformées
print(df_patient.head())
```

3. Sauvegarde dans un nouveau fichier.csv

```
# 5. Sauvegarder les données transformées dans un fichier CSV
df_patient.to_csv('C:/Users/Med Amine Salhi/Desktop/datawarehouse_salhi/datawarehouse_salhi/transformed_patient_data.csv', index=False)
print("Les données transformées ont été sauvegardées dans 'transformed_patient_data.csv'.")
```

4. Chargement des données dans la table MySQL

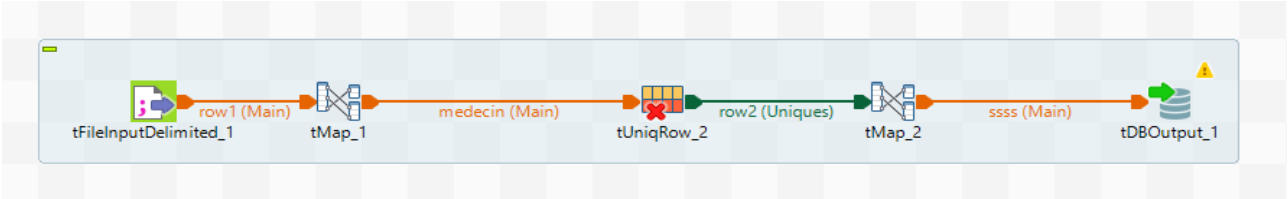
```
import pandas as pd
from sqlalchemy import create_engine

# Charger les données transformées
df_patient = pd.read_csv('C:/Users/Med Amine Salhi/Desktop/datawarehouse_salhi/patient_data.csv')

# Configuration de la connexion MySQL
username = 'root'  # Nom d'utilisateur MySQL
password = ''  #Mot de passe vide
host = 'localhost'  # Adresse du serveur MySQL
database = 'datawarehouse'  # Nom de la base de données

# Créer l'engine SQLAlchemy pour se connecter à MySQL
connection_url = f'mysql+mysqlconnector://{username}:{password}@{host}/{database}'
engine = create_engine(connection_url)

# Charger les données dans la table existante(j'ai deja creer la table patient dans Mysql )
df_patient.to_sql('patient', con=engine, if_exists='append', index=False)

print("Les données ont été ajoutées avec succès à la table 'patient'.")
```

Loading...

2.2.4 Résultat

idpatient	person_first_name	person_last_name	person_full_name	person_age_range	job	marital_status	gender	weight_range	practice_sport	smokes	alcohol
1	Javier	Rose	Javier Rose	31-40	Conservation officer historic buildings	Single	M	91+	1	0	1
2	Eric	Brown	Eric Brown	61+	Herpetologist	Married	M	71-80	1	1	0
3	April	Banks	April Banks	61+	Analytical chemist	Single	F	91+	1	1	0
4	Rick	Santana	Rick Santana	61+	Production manager	Single	M	91+	1	1	1
5	Teresa	Jones	Teresa Jones	31-40	Arts administrator	Married	F	81-90	0	0	0
6	Justin	Rivera	Justin Rivera	61+	Industrial buyer	Married	M	91+	0	1	0
7	Carolyn	Brock	Carolyn Brock	61+	Geophysicist/field seismologist	Single	F	91+	1	0	1
8	Shelley	Brown	Shelley Brown	31-40	Garment/textile technologist	Married	F	91+	0	1	1
9	Angela	Hampton	Angela Hampton	41-50	Physicist medical	Married	F	51-60	0	0	0
10	Alexis	Stanley	Alexis Stanley	61+	Customer service manager	Single	F	91+	1	1	1
11	Victoria	Anderson	Victoria Anderson	61+	Company secretary	Married	F	61-70	1	1	0
12	Jacqueline	Cobb	Jacqueline Cobb	41-50	Surveyor land/geomatics	Single	F	91+	1	0	1
13	Rebecca	Wood	Rebecca Wood	61+	Theatre stage manager	Married	F	81-90	1	0	0
14	Maria	Brown	Maria Brown	31-40	Agricultural engineer	Married	F	91+	0	1	0
15	Brandi	Johnson	Brandi Johnson	51-60	Senior tax professional/tax inspector	Single	F	91+	0	0	0
16	Ethan	Chandler	Ethan Chandler	61+	Audiological scientist	Married	M	91+	1	0	1
17	Emily	Davis	Emily Davis	51-60	Comptroller	Single	F	91+	0	0	0
18	Meagan	Gutierrez	Meagan Gutierrez	61+	Restaurant manager fast food	Single	F	91+	0	1	0
19	Thomas	Fowler	Thomas Fowler	31-40	Therapist sports	Married	M	91+	0	1	0
20	Alexis	Jackson	Alexis Jackson	19-30	Computer games developer	Single	F	91+	1	0	0
21	Rebecca	Chapman	Rebecca Chapman	61+	Advice worker	Married	F	81-90	0	1	0
22	Bobby	Valdez	Bobby Valdez	41-50	Maintenance engineer	Married	M	61-70	1	1	1

2.3 La Dimension Médecin



2.3.1 Transformation des Colonnes

Pour la dimension **Médecin**, nous avons appliqué les transformations suivantes :

1. **Création de la colonne doctor_full_name** : La colonne **doctor_full_name** est créée en combinant les colonnes **doctor_first_name** et **doctor_last_name** :

   row1.doctor_first_name + row1.doctor_last_name

2. **Transformation de l'Âge (doctor_age)** : L'âge du médecin est également transformé en utilisant la fonction **getAgeRange** :

   routine1.getAgeRange(row1.doctor_age)

3. **Ajout de la Colonne id** : De même que pour la dimension **Patient**, nous avons ajouté une colonne **id** générée automatiquement et incrémentée par 1 pour chaque enregistrement.

2.3.2 Charge dans MySQL

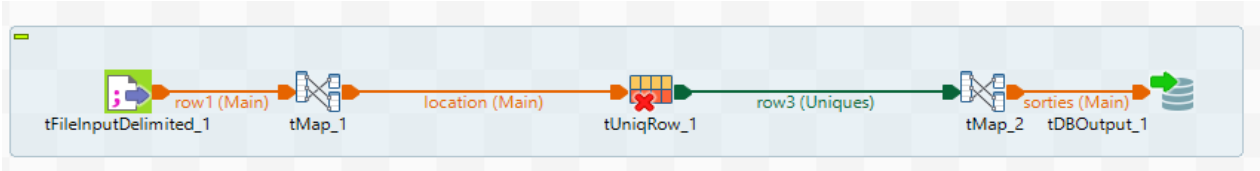
Les données de la dimension **Médecin** sont ensuite chargées dans la table **Doctor** de la base de données MySQL.

iddoctor	doctor_first_name	doctor_last_name	doctor_full_name	age_range	speciality	experience
1	Aaron	Scott	AaronScott	31-40	Neurology	10
2	Roy	Combs	RoyCombs	19-30	Pediatrics	2
3	Selena	Gregory	SelenaGregory	61+	Pediatrics	14
4	Ronald	Scott	RonaldScott	41-50	Dermatology	14
5	Robert	Scott	RobertScott	19-30	Neurology	2
6	Kim	Porter	KimPorter	51-60	Psychiatry	27
7	Tonya	Paul	TonyaPaul	31-40	Pediatrics	5
8	Debbie	Smith	DebbieSmith	61+	Dermatology	2
9	Bryan	Johnson	BryanJohnson	19-30	Cardiology	1
10	Crystal	Morris	CrystalMorris	31-40	Dermatology	1
11	James	Velasquez	JamesVelasquez	31-40	Neurology	7
12	Thomas	Hood	ThomasHood	41-50	Psychiatry	19
13	Terri	Smith	TerriSmith	51-60	Psychiatry	28
14	Dillon	Beard	DillonBeard	61+	Neurology	12
15	Nicholas	Lee	NicholasLee	61+	Cardiology	15
16	Rebecca	Harrington	RebeccaHarrington	51-60	Psychiatry	8
17	Natalie	Tucker	NatalieTucker	41-50	Dermatology	14
18	Sarah	Figueroa	SarahFigueroa	19-30	Neurology	1
19	Desiree	Watts	DesireeWatts	19-30	Cardiology	3
20	Jaime	Adams	JaimeAdams	41-50	Neurology	7
21	Karen	Sullivan	KarenSullivan	41-50	Psychiatry	3
22	Michelle	Shelton	MichelleShelton	19-30	Psychiatry	4
23	Kelly	Torres	KellyTorres	19-30	Psychiatry	3
24	Kendra	Caldwell	KendraCaldwell	41-50	Psychiatry	3
25	Jennifer	Truiillo	JenniferTruiillo	61+	Psychiatry	36

2.3.3 Code python

```python
import pandas as pd
from sqlalchemy import create_engine

df_total = pd.read_csv('C:/Users/Med Amine Salhi/Desktop/datawarehouse_salhi/tous.csv')
# Extraction des colonnes pour la dimension Médecin
df_doctor = df_total[['doctor_first_name', 'doctor_last_name', 'doctor_age','speciality','experience']]
# Création de la colonne `doctor_full_name`
df_doctor['doctor_full_name'] = df_doctor['doctor_first_name'] + ' ' + df_doctor['doctor_last_name']

# Transformation de l'âge (`doctor_age`)
def get_age_range(age):
    if age < 30:
        return 'Young'
    elif age < 50:
        return 'Middle-aged'
    else:
        return 'Senior'

df_doctor['age_range'] = df_doctor['doctor_age'].apply(get_age_range)
# Ajout de la colonne `id`
df_doctor['id'] = range(1, len(df_doctor) + 1)
#  Sauvegarder les données extraites et transformées dans un fichier CSV
df_doctor.to_csv('C:/Users/Med Amine Salhi/Desktop/datawarehouse_salhi/Projet/transformed_doctor_data.csv', index=False)
print("Les données transformées ont été sauvegardées dans 'transformed_doctor_data.csv'.")

# Configuration de la connexion MySQL
username = 'root'
password = ''
host = 'localhost'
database = 'datawarehouse'
connection_url = f'mysql+mysqlconnector://{username}:{password}@{host}/{database}'
engine = create_engine(connection_url)
# Charger les données dans la base de données MySQL
df_doctor.to_sql('doctor', con=engine, if_exists='replace', index=False)
print("Les données ont été chargées avec succès dans la table 'doctor'.")
```

2.4 La Dimension Location



2.4.1 Transformation et Granularité

Pour la dimension **Location**, aucune transformation spécifique n'a été nécessaire. Elle contient simplement les informations suivantes :

- **Hospital Name**
- **City**
- **Country**

La granularité a été respectée en maintenant les données telles qu'elles étaient fournies, sans transformation.

2.4.2 Ajout de la Colonne id

Comme pour les autres dimensions, une colonne **id** a été ajoutée à la dimension **Location**, générée automatiquement pour chaque enregistrement.

2.4.3 Charge dans MySQL

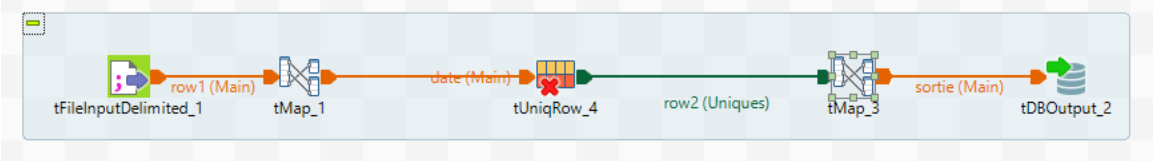
Les données de la dimension **Location** sont chargées dans la table **Location** de la base de données MySQL.

idlocation	hospital_name	city	country
1	Nguyen LLC	Paris	France
2	Freeman Woodard and Williams	Lyon	France
3	Payne and Sons	Paris	France
4	Johnson Group	Nice	France
5	Mccoy-Navarro	Birmingham	England
6	Acosta Kim and Cooper	Manchester	England
7	Chang Ltd	Leeds	England
8	Jones Inc	Lyon	France
9	Callahan Johnson and Hoffman	Liverpool	England
10	Wood and Sons	Toulouse	France
11	Webb PLC	London	England
12	Gibson Moore and Rodriguez	Lyon	France
13	Bell-Alexander	Nice	France
14	Zamora Group	London	England
15	Bryant-Martinez	Manchester	England
16	Smith PLC	Liverpool	England
17	Coleman-Herrera	Marseille	France
18	Warren-Miller	Nice	France
19	Smith PLC	London	England
20	Petersen Carrillo and Burton	Leeds	England
21	Hopkins-Boyle	London	England
22	Jennings Armstrong and Jordan	Manchester	England
23	Hayden Hicks and Robertson	Manchester	England

2.4.5 Code python

```python
import pandas as pd
from sqlalchemy import create_engine

df_total = pd.read_csv('C:/Users/Med Amine Salhi/Desktop/datawarehouse_salhi/tous.csv')
#Extraction des colonnes
df_location = df_total[['hospital_name', 'city', 'country']]

#Ajout de l'id
df_location['id'] = range(1, len(df_location) + 1)

df_location.to_csv('C:/Users/Med Amine Salhi/Desktop/datawarehouse_salhi/transformed_location_data.csv', index=False)
print("Les données transformées ont été sauvegardées dans 'transformed_location_data.csv'.")

#Configuration de la connexion MySQL
username = 'root'
password = ''
host = 'localhost'
database = 'datawarehouse'
connection_url = f'mysql+mysqlconnector://{username}:{password}@{host}/{database}'
engine = create_engine(connection_url)

#Charger les données dans la base de données MySQL
df_location.to_sql('locationdimension', con=engine, if_exists='replace', index=False)
```

2.5 La Dimension Temps (Date)



2.5.1 Transformation de la Date

Pour la dimension **Date**, nous avons extrait différentes parties de la date à l'aide de la fonction Talend **TalendDate.getPartOfDate**. Cela nous a permis de décomposer la date en plusieurs granularités :

1. **Année** :

   TalendDate.getPartOfDate("YEAR", row1.date)

2. **Mois** :

   TalendDate.getPartOfDate("MONTH", row1.date) + 1

on ajouter +1 puisque la fonction commence de 0->11.

3. **Jour de la semaine** :

   TalendDate.getPartOfDate("DAY_OF_WEEK", row1.date)

2.5.2 Ajout de la Colonne id

Une fois les différentes parties de la date extraites, une colonne **iddate** a été ajoutée pour chaque ligne, générée automatiquement et incrémentée.

2.5.3 Charge dans MySQL

Les données de la dimension **Date** sont ensuite chargées dans la table **Date** de la base de données MySQL, avec respect de la granularité **Jour/Mois/Année**.

iddate	date	day	month	year
1	2022-12-22	5	12	2022
2	2020-12-24	5	12	2020
3	2022-05-19	5	5	2022
4	2021-07-02	6	7	2021
5	2022-09-05	2	9	2022
6	2022-02-07	2	2	2022
7	2021-09-03	6	9	2021
8	2022-09-26	2	9	2022
9	2022-06-29	4	6	2022
10	2021-06-20	1	6	2021
11	2022-12-02	6	12	2022
12	2021-04-11	1	4	2021

2.5.4 Code python

```python
import pandas as pd
from sqlalchemy import create_engine
df_total = pd.read_csv('C:/Users/Med Amine Salhi/Desktop/datawarehouse_salhi/tous.csv')

#Extraction de la colone date
df_date = df_total[['date']]

#Transformation des données
# Extraction de l'année, du mois et du jour de la semaine
df_date['year'] = df_date['date'].dt.year
df_date['month'] = df_date['date'].dt.month +1 #j'ai ajoute 1 puisque [0,11]
df_date['day'] = df_date['date'].dt.day

#Ajout de la colonne `id`
df_date['id'] = range(1, len(df_date) + 1)

# Garder uniquement les colonnes pertinentes
df_date = df_date[['id', 'date', 'year', 'month', 'day']]

# Sauvegarder les données extraites et transformées dans un fichier CSV (optionnel)
df_date.to_csv('C:/Users/Med Amine Salhi/Desktop/datawarehouse_salhi/transformed_date_data.csv', index=False)
print("Les données transformées ont été sauvegardées dans 'transformed_date_data.csv'.")

# 5. Configuration de la connexion MySQL
username = 'root'
password = ''
host = 'localhost'
database = 'datawarehouse'
connection_url = f'mysql+mysqlconnector://{username}:{password}@{host}/{database}'
engine = create_engine(connection_url)

# 6. Charger les données dans la base de données MySQL
df_date.to_sql('datedimension', con=engine, if_exists='replace', index=False)
print("Les données ont été chargées avec succès dans la table 'datedimension'.")
```

2.6 Utilisation de tUniqRow dans Talend

Dans Talend, nous avons utilisé la fonction **tUniqRow** pour garantir que chaque ligne dans les dimensions soit unique. Cette fonctionnalité joue un rôle clé dans l'élimination des doublons en vérifiant que chaque enregistrement reste distinct, assurant ainsi l'intégrité des données lors de l'importation dans MySQL.

- **Fonction de tUniqRow :** Cette fonction permet de garantir que les lignes traitées ne soient pas dupliquées avant leur chargement dans la base de données, ce qui est essentiel pour assurer la qualité des données dans l'entrepôt.

Conclusion

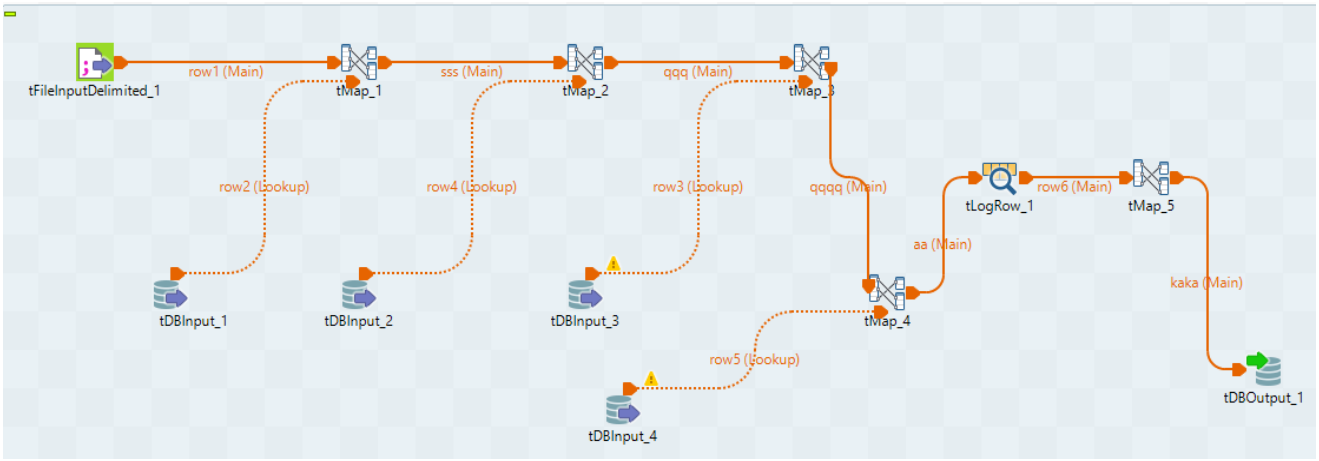
Dans cette étape, nous avons extrait et transformé les quatre dimensions du dataset, tout en respectant les règles de granularité et d'intégrité des données. Les données traitées ont ensuite été chargées dans la base de données **MySQL**, prêtes à être utilisées pour des analyses futures.

Étape 3 : Création de la Table des Faits Consultation

3.1 Objectif de l'Étape

L'objectif de cette étape était de **créer la table des faits Consultation** en utilisant les données provenant de plusieurs dimensions (Patient, Médecin, Location, Temps). Comme les **clés primaires** de chaque dimension étaient déjà générées et stockées dans MySQL, nous avons utilisé **Talend** pour **faire la jointure** entre les données de **tous.csv** et les dimensions, puis avons créé une **table des faits** avec les **clés étrangères** correspondantes.

3.2 Processus dans Talend



Dans **Talend**, nous avons suivi les étapes suivantes pour créer la table des faits **Consultation** :

1. **Chargement des fichiers source :**
    - Le fichier **tous.csv** contenant toutes les données de notre dataset a été chargé dans Talend via un composant **tFileInputDelimited**. Ce fichier contient des informations brutes, mais ne comprend pas les clés primaires des dimensions.
2. **Chargement des Dimensions :**
    - Chaque **dimension (Patient, Médecin, Location, Temps)** a été chargée dans Talend à partir de la base de données MySQL où les **clés primaires** avaient été préalablement stockées. Nous avons utilisé un composant **tMysqlInput** pour extraire les données de chaque dimension.
    - Par exemple, la **dimension Patient** contient des informations comme le nom complet, l'âge, le poids, et l'ID généré dans la base de données.

3. **Création des Clés Primaires et Étrangères :**

    - Les **clés primaires** dans les dimensions étaient déjà définies dans MySQL, donc l'objectif était d'utiliser ces **clés primaires** pour faire les **jointures** avec les enregistrements de **tous.csv** et générer les **clés étrangères**.

    - Par exemple, dans le fichier **tous.csv**, le nom du médecin (**doctor_first_name**, **doctor_last_name**) a été utilisé pour **faire correspondre** les données avec la **dimension Médecin** et récupérer l'**ID_doctor** comme clé étrangère.

4. **Jointure des Dimensions avec tous.csv :**

    - Nous avons utilisé le composant **tMap** de Talend pour effectuer des **jointures internes** (inner join) entre le fichier **tous.csv** et les dimensions. Par exemple :

        - La colonne **doctor_name** dans **tous.csv** a été jointe avec les colonnes **doctor_first_name** et **doctor_last_name** dans la dimension **Médecin** pour récupérer l'**ID_doctor**.

        - La même chose a été faite pour les autres dimensions (Patient, Location, Temps) en utilisant des jointures sur les colonnes correspondantes.

5. **Création de la Table des Faits :**

    - Après avoir récupéré les **clés étrangères** des différentes dimensions, nous avons créé la **table des faits Consultation**. Cette table contient les clés étrangères des dimensions (**ID_patient**, **ID_doctor**, **ID_location**, **ID_time**) ainsi que les mesures comme le **prix** et la **durée** des consultations.

    - La **table des faits** a été alimentée avec les informations extraites et les **clés étrangères** ajoutées.

6. **Chargement de la Table des Faits dans MySQL :**

    - Une fois la **table des faits Consultation** créée, les données ont été insérées dans la table **consultation** de la base de données **MySQL** en utilisant un composant **tMysqlOutput**.

    - La table des faits a été chargée dans MySQL avec les **clés étrangères** et les mesures agrégées (prix, durée).

3.3 Composants utilisés dans Talend

- **tFileInputDelimited** : Pour charger les données à partir du fichier **tous.csv**.
- **tMysqlInput** : Pour extraire les données des dimensions stockées dans MySQL.
- **tMap** : Pour effectuer les jointures entre le fichier **tous.csv** et les dimensions, et créer la table des faits.
- **tMysqlOutput** : Pour charger la table des faits **Consultation** dans MySQL.

3.4 Résultats de l'Étape

Une fois toutes les jointures effectuées, nous avons obtenu une **table des faits Consultation** avec les **clés étrangères** des dimensions et les mesures associées aux consultations. Cette table a été chargée dans la base de données MySQL et est désormais prête à être utilisée pour les analyses futures.

id	iddoctor	idpatient	iddate	idlocation	price	duration
2	116	1359	1	1	107	108
3	159	2	2	2	56	48
8	154	3	3	3	112	78
9	134	4	4	4	121	114
11	104	5	5	5	20	36
15	147	1001	6	6	117	12
16	184	1270	7	7	40	42
19	117	1088	8	8	58	72
21	156	9	9	9	108	84
23	193	10	10	10	118	78
24	113	1479	11	11	44	114
25	157	1305	12	12	154	60

Code Python pour la même opération

```python
import pandas as pd
from sqlalchemy import create_engine

# 1. Charger le fichier contenant les données brutes
df_total = pd.read_csv('C:/Users/Med Amine Salhi/Desktop/datawarehouse_salhi/tous.csv')

# 2. Connexion à la base de données MySQL pour récupérer les clés primaires des dimensions
username = 'root'
password = ''
host = 'localhost'
database = 'datawarehouse'
engine = create_engine(f'mysql+mysqlconnector://{username}:{password}@{host}/{database}')

# 3. Récupérer les clés primaires des dimensions depuis MySQL
df_patient = pd.read_sql("SELECT * FROM patient", engine)
df_doctor = pd.read_sql("SELECT * FROM doctor", engine)
df_location = pd.read_sql("SELECT * FROM location", engine)
df_time = pd.read_sql("SELECT * FROM time", engine)

# 4. Faire la jointure entre `tous.csv` et les dimensions sur les colonnes correspondantes
# Jointure pour obtenir la clé primaire des dimensions dans le fichier `tous.csv`

# Jointure avec la dimension Patient (exemple : patient_name dans `tous.csv` correspond à person_full_name dans la table Patient)
df_facts = df_total.merge(df_patient[['idpatient', 'person_first_name']], how='left', left_on='patient_first_name', right_on='person_first_name')

# Jointure avec la dimension Médecin
df_facts = df_facts.merge(df_doctor[['iddoctor', 'doctor_full_name']], how='left', left_on='doctor_name', right_on='doctor_full_name')

# Jointure avec la dimension Location
df_facts = df_facts.merge(df_location[['idlocation', 'hospital_name']], how='left', left_on='location', right_on='hospital_name')

# Jointure avec la dimension Temps
df_facts = df_facts.merge(df_time[['iddate', 'date']], how='left', left_on='date', right_on='date')

# 5.Creation de la table de fait
df_facts = df_facts[['iddoctor','idpatient','iddate','idlocation','price','duration']]

# 6. Charger la table des faits dans MySQL
df_facts.to_sql('consultation', con=engine, if_exists='replace', index=False)

print("La table des faits 'Consultation' a été créée et chargée dans MySQL.")
```

Conclusion

Cette étape dans Talend vous a permis de créer la **table des faits Consultation** en reliant les données de **tous.csv** aux dimensions existantes via des **jointures internes**. Vous avez ensuite chargé cette table dans MySQL, et elle est maintenant prête pour des analyses futures.

Étape 4 : Création et Déploiement du Cube OLAP

4.1 Introduction

Le cube OLAP (Online Analytical Processing) est une structure multidimensionnelle utilisée pour analyser rapidement de grandes quantités de données. Il permet des explorations dynamiques et des calculs d'agrégations pré-calculées, offrant ainsi des performances optimisées pour des analyses complexes. L'objectif de cette étape est de créer un cube OLAP basé sur les données transférées depuis MySQL vers SQL Server Analysis Services (SSAS).

4.2 Préparation des Données

Les données de MySQL ont été extraites et chargées dans SSAS en utilisant **Microsoft SQL Server Migration Assistant for MySQL**, garantissant leur conformité avec les exigences de SSAS. Les relations entre la table des faits (Consultation) et les dimensions (Patient, Médecin, Temps, Location) ont été établies pour assurer une modélisation correcte.

4.3 Création du Cube OLAP

4.3.1 Initialisation du Projet

Un projet multidimensionnel a été créé dans **Visual Studio** à l'aide de l'extension **Microsoft Analysis Services Projects**.

4.3.2 Configuration de la Source de Données

1. Création de la Source de Données :
   Un fichier de source de données a été créé dans **Visual Studio** pour permettre à **SSAS** d'accéder aux tables nécessaires à la création du cube OLAP.

2. Choix du Serveur SQL Server :
   Le serveur **SQL Server** a été sélectionné pour héberger le cube, en garantissant la compatibilité avec les outils d'analyse.

3. Authentification :
   L'authentification a été configurée via **SQL Server Authentication**, selon les besoins, et la connexion a été testée pour assurer son bon fonctionnement.

4.3.3 Modélisation des Dimensions

Les dimensions ont été définies avec leurs attributs clés, par exemple :

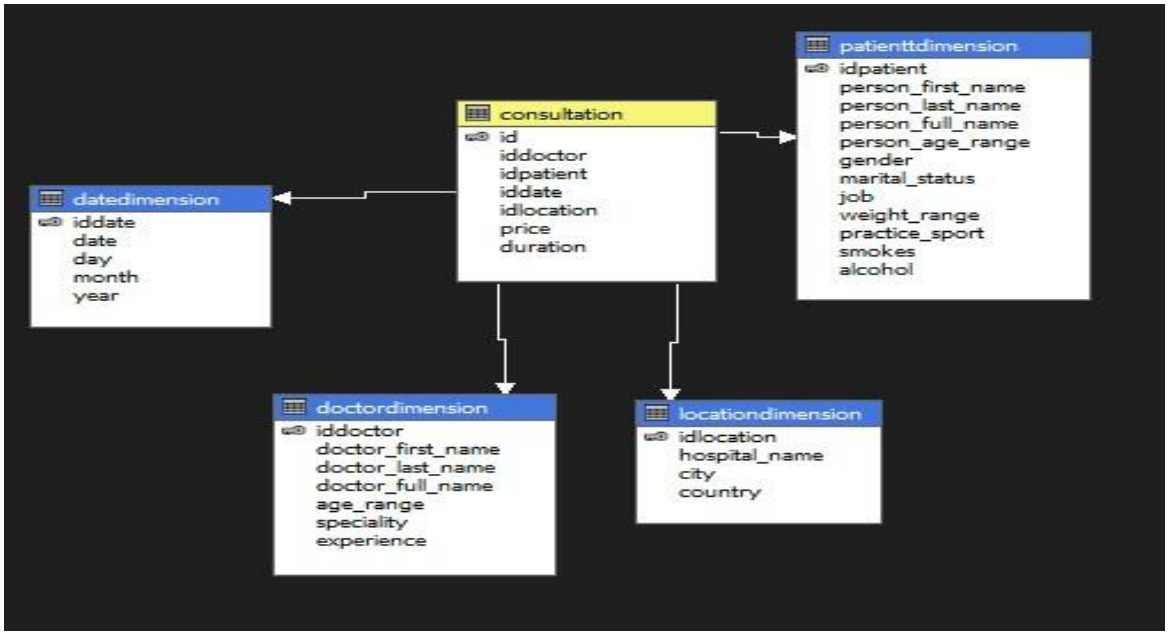
- **Dimension Patient :** Attributs tels que idpatient, age, sexe, poids**.**
- **Dimension Temps :** Hiérarchie temporelle (Année → Mois → Jour).
- **Dimension Médecin :** Attributs comme idmedecin, nommedecin, specialite.

343.4 Définition des Mesures

Des mesures comme le **nombre de consultations** et les **revenus (Price)** ont été créées à partir de la table des faits.

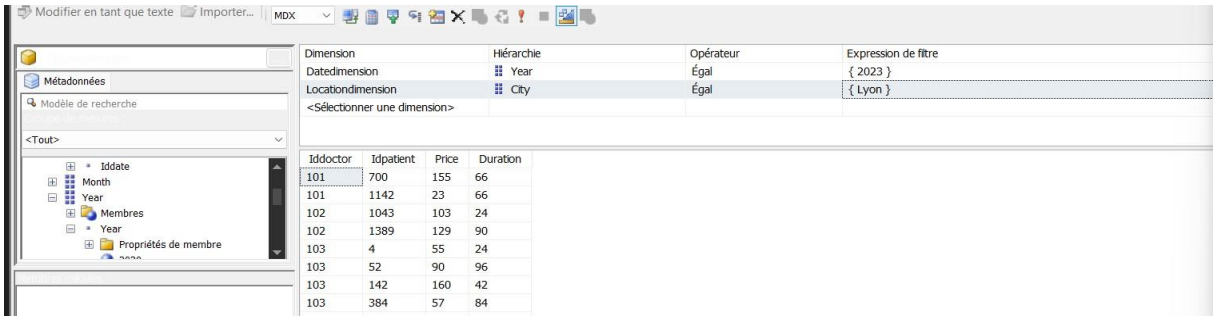
4.3.5 Construction du Cube

Le cube a été construit et déployé sur SSAS, avec un traitement pour générer les agrégations nécessaires à l'analyse rapide des données.



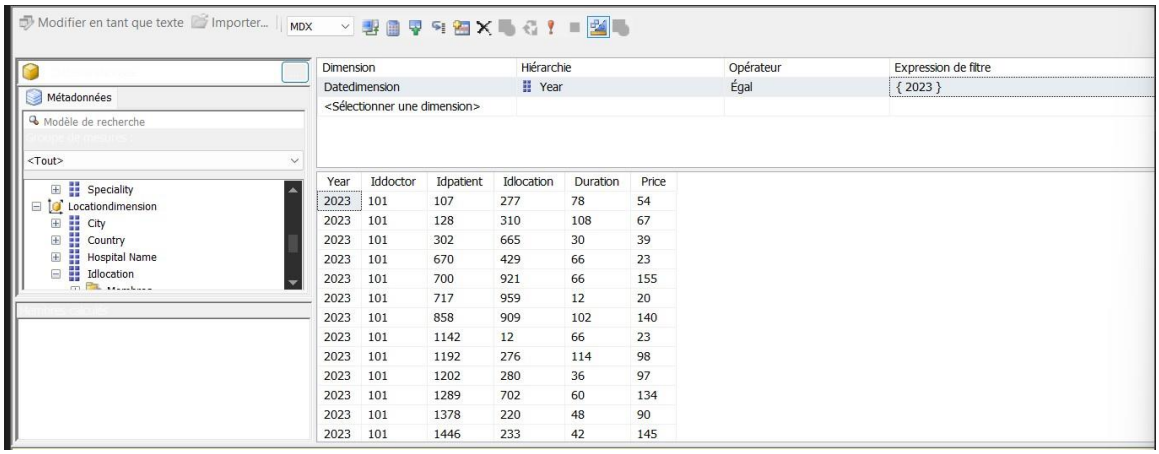
4.4 Opérations d'Analyse du Cube

1. **Slice (Découpage)** : Analyse des données en fonction d'une dimension spécifique, comme les consultations par mois.




```
SELECT NON EMPTY { [Measures].[Price], [Measures].[Duration] } ON COLUMNS, NON EMPTY { ([Doctordimension].[Iddoctor].[Iddoctor].ALLMEMBERS * [Patienttdimension].[Idpatient].[Idpatient].A
DIMENSION PROPERTIES MEMBER_CAPTION, MEMBER_UNIQUE_NAME ON ROWS FROM ( SELECT ( { [Locationdimension].[City].&[Lyon] } ) ON COLUMNS FROM ( SELECT ( { [Datedimension].[Year]
FROM [Datawarehousee])) WHERE ( [Datedimension].[Year].&[2023], [Locationdimension].[City].&[Lyon] ) CELL PROPERTIES VALUE, BACK_COLOR, FORE_COLOR, FORMATTED_VALUE, FORMAT_S
FONT_SIZE, FONT_FLAGS
```

2. **Dice (Exploration)** : Exploration croisée de plusieurs dimensions pour identifier des relations, par exemple, les revenus par médecin et hôpital.



Year	Iddoctor	Idpatient	Idlocation	Duration	Price
2023	101	107	277	78	54
2023	101	128	310	108	67
2023	101	302	665	30	39
2023	101	670	429	66	23
2023	101	700	921	66	155
2023	101	717	959	12	20
2023	101	858	909	102	140
2023	101	1142	12	66	23
2023	101	1192	276	114	98
2023	101	1202	280	36	97
2023	101	1289	702	60	134
2023	101	1378	220	48	90
2023	101	1446	233	42	145

```
SELECT NON EMPTY { [Measures].[Duration], [Measures].[Price] } ON COLUMNS, NON EMPTY { ([Datedimension].[Year].[Year].ALLMEMBERS * [Doctordimension].[Iddoctor].[Iddoctor].ALLMEMBER
[Idpatient].[Idpatient].ALLMEMBERS * [Locationdimension].[Idlocation].[Idlocation].ALLMEMBERS ) } DIMENSION PROPERTIES MEMBER_CAPTION, MEMBER_UNIQUE_NAME ON ROWS FROM ( SELEC
[Year].&[2023] } ) ON COLUMNS FROM [Datawarehousee]) CELL PROPERTIES VALUE, BACK_COLOR, FORE_COLOR, FORMATTED_VALUE, FORMAT_STRING, FONT_NAME, FONT_SIZE, FONT_FLAGS
```

---

3. **Agrégations** : Calculs automatiques des moyennes, totaux, et autres indicateurs clés.

```
SELECT
    NON EMPTY { [Measures].[Price] } ON COLUMNS,
    NON EMPTY {
        ([Doctordimension].[Doctor Full Name].[Doctor Full Name].ALLMEMBERS)
    } ON ROWS
FROM [Datawarehousee]
CELL PROPERTIES VALUE, BACK_COLOR, FORE_COLOR, FORMATTED_VALUE, FORMAT_STRING, FONT_NAME, FONT_SIZE, FONT_FLAGS
```

Doctor Full Name	Price
Aaron Scott	4648
Alison Sampson	4750
Alyssa Cox	4199
Andrea Padilla	4845
Andres Banks	4487
April Gray	4527
Ashlee Ramirez	4745

4.5 Importance du Cube OLAP

Le cube OLAP permet :

- **Des performances optimisées** grâce aux agrégations pré-calculées.
- **Une exploration multidimensionnelle** flexible, facilitant l'analyse sous différents angles.

- **L’intégration avec des outils comme Power BI et Excel** pour une analyse collaborative.

Conclusion

Le cube OLAP créé avec SSAS permet d'optimiser l'analyse des données en offrant une solution flexible et performante. Son intégration avec Power BI et Excel facilite la création de rapports dynamiques et la collaboration autour des données.

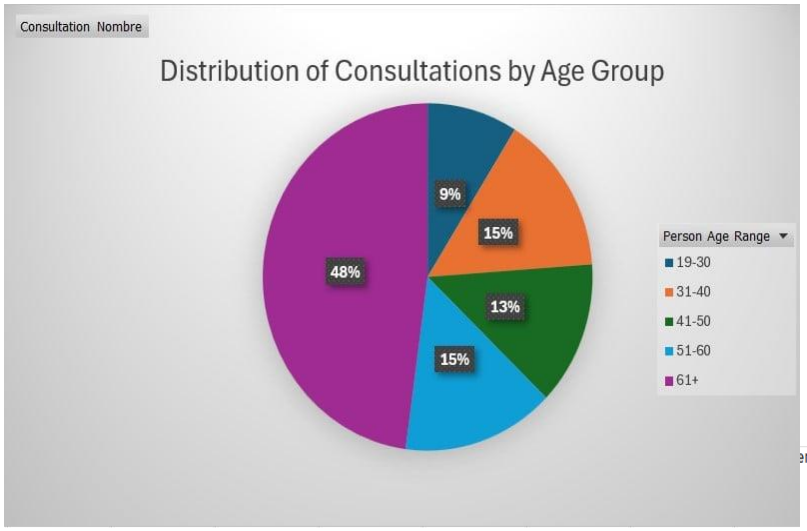
Étape 5 : Analyse des Données avec Power BI et Excel

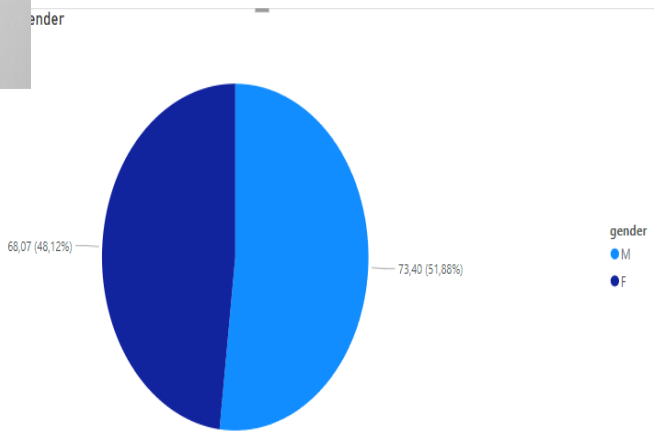
5.1 Introduction

Une fois le cube OLAP créé, il a été utilisé dans **Power BI** et **Excel** pour effectuer des analyses interactives et dynamiques. Ces outils ont permis d'explorer en profondeur les données multidimensionnelles et de générer des rapports détaillés sur les consultations, les revenus et la performance des médecins.

5.2 Analyse avec Power BI

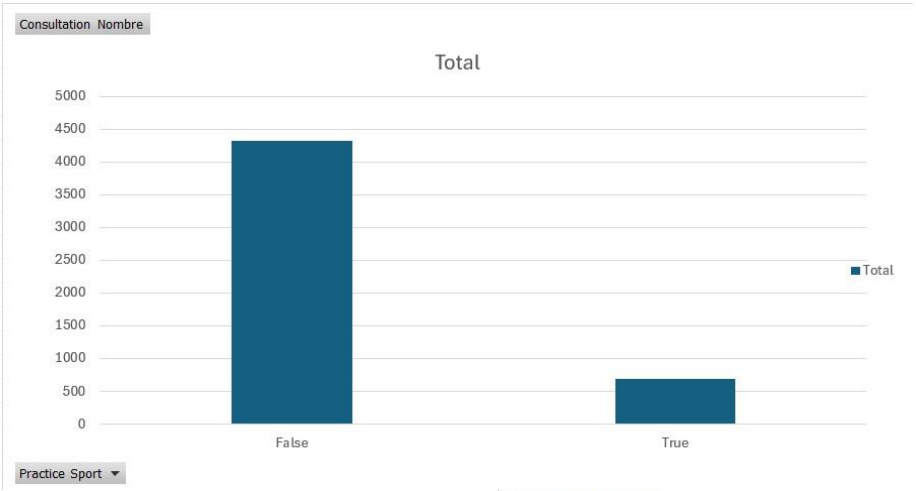
Power BI a été connecté au cube OLAP pour créer des rapports interactifs. Des visualisations telles que des **graphes en barres**, **diagrammes circulaires** et **courbes de tendance** ont été utilisées :

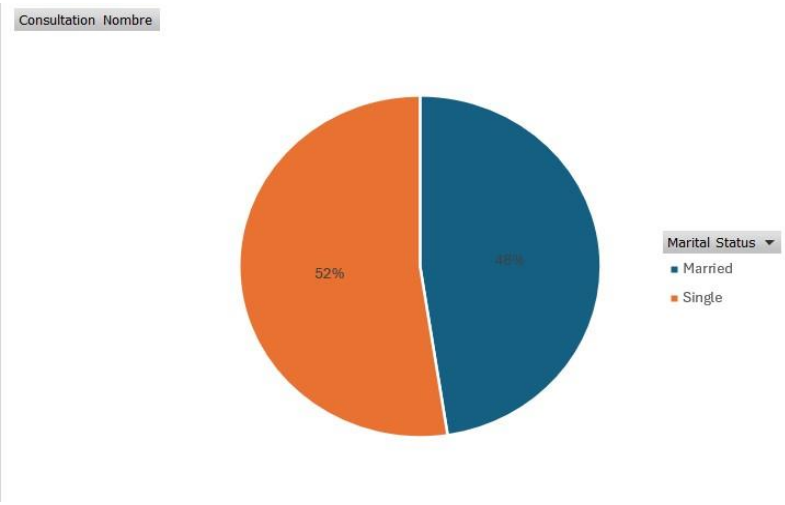




4.3 Analyse avec Excel

Excel a été utilisé pour se connecter au cube OLAP et effectuer des **tableaux croisés dynamiques**. Cela a permis d'analyser des données telles que :





Conclusion Finale

Ce projet a permis de développer un **entrepôt de données** solide et fonctionnel en suivant une approche structurée et méthodique, en commençant par la **conception de l'entrepôt de données** jusqu'à la création d'un **cube OLAP** et l'analyse approfondie des données. Chaque étape a contribué à transformer des données brutes en informations stratégiques et exploitées de manière optimale.

Récapitulatif des Étapes Clés :

1. **Conception de l'Entrepôt de Données (Étape 0)** :
    - L'analyse des besoins a permis de définir les dimensions essentielles (Patient, Médecin, Location, Temps) et d'établir la structure de l'entrepôt. Le modèle en **étoile** a été choisi pour sa simplicité et son efficacité dans les analyses.
2. **Création et Chargement des Dimensions et Table de Faits (Étape 1 et 2)** :
    - Les dimensions et la table de faits ont été extraites, filtrées et nettoyées à partir des fichiers sources (CSV, JSON, Access), et chargées dans la base de données MySQL. Nous avons ensuite mis en place les relations entre les différentes tables et leur correspondance avec les clés primaires et étrangères.
3. **Création du Cube OLAP (Étape 3)** :
    - Nous avons utilisé **SQL Server Analysis Services (SSAS)** pour construire un cube OLAP multidimensionnel, qui a permis de définir les dimensions, mesures et hiérarchies. Ce cube a été déployé et traité pour effectuer des analyses rapides grâce à des agrégations pré-calculées.
4. **Analyse des Données avec Power BI et Excel (Étape 4)** :
    - **Power BI** a été utilisé pour créer des rapports dynamiques et des visualisations interactives, permettant d'analyser les consultations, les revenus, et la performance des médecins. **Excel** a également été utilisé pour explorer le cube à travers des tableaux croisés dynamiques, facilitant des analyses ad hoc et détaillées.

Impact et Résultats :

Le cube OLAP, en combinaison avec des outils comme **Power BI** et **Excel**, a permis d'obtenir des **insights stratégiques** pour améliorer la gestion des ressources médicales, optimiser les plannings et mieux comprendre les comportements des patients. Les analyses ont permis de détecter des tendances cruciales, telles que la **répartition des consultations** par spécialité, les **revenus par hôpital**, et la **performance des médecins**.

Ce projet a donc non seulement répondu aux objectifs définis, mais a également fourni une **solution flexible et évolutive** pour des analyses futures. Les outils mis en place permettent désormais une exploration facile et rapide des données, et ouvrent la voie à d'autres améliorations dans la gestion des informations médicales.