Rapport sur le Projet de Data Warehouse Météorologique

Étudiants:
AIT MOKHTAR Mohamed Amine
MOKHTARI Dhia El Hak
BENSEGHIR Leila Siham

June 3, 2024

1 Introduction

Ce rapport présente le développement d'un entrepôt de données pour les données climatiques de trois pays du Maghreb (Algérie, Maroc et Tunisie). L'entrepôt de données permet un stockage, une récupération et une analyse efficaces des données météorologiques. La mise en œuvre utilise Python pour l'extraction, la transformation et le chargement des données (ETL), et MySQL pour la gestion de la base de données. De plus, un tableau de bord créé à l'aide du framework Dash fournit des visualisations dynamiques des données climatiques.

2 Schéma en Étoile de l'Entrepôt de Données

Le schéma en étoile conçu pour l'entrepôt de données météorologiques se compose des tables suivantes :

2.1 Table de Faits : `Weather_Fact`

- **StationID** : Identifiant de la station météorologique (clé étrangère vers `Station_Dim`).
- **Date_ID** : Identifiant de la date (clé étrangère vers `Date_Dim`).
- **PRCP** : Précipitations (mm).
- **TAVG** : Température moyenne (°C).
- **TMAX** : Température maximale (°C).
- **TMIN** : Température minimale (°C).
- **SNWD** : Profondeur de neige (mm).
- **PGTM** : Heure de la rafale maximale (hPa).
- **SNOW** : Chute de neige (mm).
- **WDFG** : Direction de la rafale maximale (°).
- **WSFG** : Vitesse maximale de la rafale (km/h).

2.2 Tables de Dimensions

2.2.1 `Station_Dim`

- **StationID** : Identifiant auto-incrémenté.
- **StationCode** : Code de la station météorologique.
- **Name** : Nom de la station météorologique.
- **Latitude** : Latitude de la station.

- **Longitude** : Longitude de la station.
- **Elevation** : Élévation de la station.
- **Pays** : Code du pays.

2.2.2 `Date_Dim`

- **Date_ID** : Identifiant auto-incrémenté.
- **Date** : Date de la mesure.
- **Year** : Année de la mesure.
- **Month** : Mois de la mesure.
- **Day** : Jour de la mesure.

3 Processus Géré par l'Entrepôt de Données

L'entrepôt de données traite les tâches suivantes :

3.1 Extraction de Données

- Les données climatiques sont extraites des fichiers CSV pour l'Algérie, le Maroc et la Tunisie.
- Les données de la dimension temporelle sont extraites d'un fichier CSV séparé (`Dim_Date_1850-2050.csv`).

3.2 Transformation de Données

- Le nettoyage des données inclut la gestion des valeurs manquantes, le formatage des dates et l'extraction de champs supplémentaires tels que l'année, le mois et le jour à partir de la date.
- La standardisation des données assure la cohérence des types et des valeurs des données.

3.3 Chargement des Données

- Les données nettoyées sont chargées dans les tables respectives (`Weather_Fact`, `Station_Dim` et `Date_Dim`) dans la base de données MySQL.

4 Granularité de la Table de Faits

La granularité de la table `Weather_Fact` est au niveau des mesures quotidiennes pour chaque station météorologique. Chaque enregistrement représente les données météorologiques collectées à partir d'une station spécifique à une date spécifique.

5 Mesures

Les principales mesures dans la table `Weather_Fact` incluent :

- Précipitations (PRCP)
- Température moyenne (TAVG)
- Température maximale (TMAX)
- Température minimale (TMIN)
- Profondeur de neige (SNWD)
- Heure de la rafale maximale (PGTM)
- Chute de neige (SNOW)
- Direction de la rafale maximale (WDFG)
- Vitesse maximale de la rafale (WSFG)

6 Dimensions

Les dimensions de l'entrepôt de données sont représentées par les tables `Station_Dim` et `Date_Dim`. Ces dimensions fournissent des informations contextuelles sur les mesures météorologiques, telles que l'emplacement de la station météorologique et la date de la mesure.

7 Tableau de Bord

Le tableau de bord, mis en œuvre à l'aide du framework Dash, fournit des visualisations dynamiques pour analyser les données climatiques. Les principales fonctionnalités du tableau de bord incluent :

7.1 Carte Géographique

Une carte visualisant différentes mesures avec des filtres sur l'année, la saison, le trimestre et le mois. Les utilisateurs peuvent sélectionner les filtres souhaités pour afficher les mesures sur la carte.

7.2 Graphiques de Séries Temporelles

Courbes retraçant l'évolution de la température, des précipitations et de la profondeur de neige sur plusieurs années. Ces graphiques fournissent des informations sur les tendances et les modèles des données climatiques.

7.3 Filtres Dynamiques

Des menus déroulants permettent aux utilisateurs de filtrer les données par station, année, saison, trimestre, mois et mesure. Cette interactivité améliore l'expérience utilisateur et offre une vue personnalisable des données.

7.4 Visualisations d'Exemple

- **Mesure Météorologique au Fil du Temps** : Un graphique linéaire montrant la mesure météorologique sélectionnée au cours des jours d'un mois sélectionné.
- **Distribution de la Mesure Météorologique** : Un histogramme affichant la distribution de la mesure météorologique sélectionnée pour les filtres spécifiés.

8 Détails de la Mise en Œuvre

8.1 Processus ETL

Le processus ETL (Extract, Transform, Load) est mis en œuvre dans le script `ETL.py`. Les principales fonctions et leurs objectifs sont :

8.1.1 `merge_files(file_path_pattern)`

- Extrait les données de plusieurs fichiers CSV correspondant au modèle de chemin de fichier donné.
- Fusionne les données en un seul DataFrame.
- Affiche le début du DataFrame et le nombre de valeurs nulles pour inspection.

```
import pandas as pd
import glob

def merge_files(file_path_pattern):
    files = glob.glob(file_path_pattern)
    print("** Merging files from {file_path_pattern} **")
    merged_df = pd.concat(map(lambda file: pd.read_csv(file, low_memory=False), files), igno
    print(merged_df.head())
    print(merged_df.isnull().sum())
    return merged_df
```

8.1.2 clean_data(df)

- Sélectionne les colonnes pertinentes et formate la colonne de date.
- Ajoute des champs supplémentaires pour l'année, le mois, le jour et le code du pays.
- Remplace des valeurs erronées spécifiques par `NaN`.
- Remplit les valeurs manquantes avec des zéros pour certaines colonnes et des valeurs médianes pour d'autres.
- Retourne le DataFrame nettoyé.

```
def clean_data(df):
    keep_columns = ['STATION', 'NAME', 'LATITUDE', 'LONGITUDE', 'ELEVATION', 'DATE', 'PRCP',
                    'TMAX', 'TMIN', 'SNWD', 'PGTM', 'SNOW', 'WDFG', 'WSFG']
    df

 = df[keep_columns]

    df['DATE'] = pd.to_datetime(df['DATE'], format='%Y-%m-%d')
    df['YEAR'] = df['DATE'].dt.year
    df['MONTH'] = df['DATE'].dt.month
    df['DAY'] = df['DATE'].dt.day
    df['PAYS'] = df['STATION'].str[:2]

    for col in ['TMIN', 'TMAX', 'TAVG', 'PGTM', 'WDFG', 'WSFG']:
        df[col] = df[col].replace([-99, 93.2], pd.NA)

    df.fillna({'PRCP': 0, 'SNWD': 0, 'SNOW': 0}, inplace=True)

    numeric_cols = df.select_dtypes(include=['float64', 'int64']).columns
    df[numeric_cols] = df[numeric_cols].fillna(df[numeric_cols].median())

    return df
```

8.1.3 main()

- Spécifie les modèles de chemin de fichier pour les fichiers de données provenant d'Algérie, du Maroc et de Tunisie.
- Fusionne et nettoie les données des trois pays.
- Enregistre les données nettoyées dans un fichier CSV.

```
def main():
    algeria_pattern = "./Dataset/Weather Data/Algeria/Weather_*.csv"
    morocco_pattern = "./Dataset/Weather Data/Morocco/Weather_*.csv"
```

```
    tunisia_pattern = "./Dataset/Weather Data/Tunisia/Weather_*.csv"

    df_algeria = merge_files(algeria_pattern)
    df_morocco = merge_files(morocco_pattern)
    df_tunisia = merge_files(tunisia_pattern)

    df = pd.concat([df_algeria, df_morocco, df_tunisia], ignore_index=True)
    print("*** Merging multiple country datasets into a single dataframe ***")
    print(df.head())

    cleaned_df = clean_data(df)
    cleaned_df.to_csv("./Dataset/Weather_data.csv", index=False)
    print("Data cleaned and saved to ./Dataset/Weather_data.csv")

if __name__ == "__main__":
    main()
```

8.2 Création et Population de la Base de Données

La création et la population de la base de données sont gérées dans le script `DB.py`. Les principales fonctions et leurs objectifs sont :

8.2.1 `create_database(connection, db_name)`

- Crée une nouvelle base de données si elle n'existe pas déjà.

```
def create_database(connection, db_name):
    cursor = connection.cursor()
    cursor.execute(f"CREATE DATABASE IF NOT EXISTS {db_name}")
    print(f"Database {db_name} created")
    cursor.close()
```

8.2.2 `create_tables(cursor)`

- Crée les tables `Station_Dim`, `Date_Dim` et `Weather_Fact` dans la base de données.

```
def create_tables(cursor):
    table_creation_queries = {
        "Station_Dim": """
            CREATE TABLE IF NOT EXISTS Station_Dim (
                StationID INT AUTO_INCREMENT,
                StationCode VARCHAR(255),
                Name VARCHAR(255),
                Latitude FLOAT,
                Longitude FLOAT,
                Elevation FLOAT,
```

```
            Pays CHAR(2),
            PRIMARY KEY (StationID)
        )
    """,
    "Date_Dim": """
        CREATE TABLE IF NOT EXISTS Date_Dim (
            Date_ID INT AUTO_INCREMENT,
            Date DATE,
            Year INT,
            Month INT,
            Day INT,
            PRIMARY KEY (Date_ID)
        )
    """,
    "Weather_Fact": """
        CREATE TABLE IF NOT EXISTS Weather_Fact (
            StationID INT,
            Date_ID INT,
            PRCP FLOAT,
            TAVG FLOAT,
            TMAX FLOAT,
            TMIN FLOAT,
            SNWD FLOAT,
            PGTM FLOAT,
            SNOW FLOAT,
            WDFG FLOAT,
            WSFG FLOAT,
            PRIMARY KEY (StationID, Date_ID),
            FOREIGN KEY (StationID) REFERENCES Station_Dim(StationID),
            FOREIGN KEY (Date_ID) REFERENCES Date_Dim(Date_ID)
        )
    """
}

for table, query in table_creation_queries.items():
    cursor.execute(query)
    print(f"Table {table} created")
```

8.2.3 `populate_tables(cursor, data)`

- Insère les données dans les tables `Station_Dim` et `Date_Dim`.
- Remplit la table `Weather_Fact` avec les données nettoyées.

```
def populate_tables(cursor, data):
    stations = data[["STATION", "NAME", "LATITUDE", "LONGITUDE", "ELEVATION", "PAYS"]].drop_
    dates = data[["DATE", "YEAR", "MONTH", "DAY"]].drop_duplicates()
```

```
    station_id_map = {}
    for _, row in stations.iterrows():
        cursor.execute("""
            INSERT INTO Station_Dim (StationCode, Name, Latitude, Longitude, Elevation, Pays
            VALUES (%s, %s, %s, %s, %s, %s)
        """, tuple(row))
        station_id_map[row["STATION"]] = cursor.lastrowid

    date_id_map = {}
    for _, row in dates.iterrows():
        cursor.execute("""
            INSERT INTO Date_Dim (Date, Year, Month, Day)
            VALUES (%s, %s, %s, %s)
        """, tuple(row))
        date_id_map[row["DATE"]] = cursor.lastrowid

    data = data.replace({np.nan: None})  # Remplace NaN par None

    for _, row in data.iterrows():
        cursor.execute("""
            INSERT INTO Weather_Fact (StationID, Date_ID, PRCP, TAVG, TMAX, TMIN, SNWD, PGTM
            VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
        """, (
        station_id_map[row["STATION"]], date_id_map[row["DATE"]], row["PRCP"], row["TAVG"],
        row["SNWD"], row["PGTM"], row["SNOW"], row["WDFG"], row["WSFG"]))

    print("Tables populated")
```

8.2.4 `populate_date_dim(cursor)`

- Remplit la table `Date_Dim` avec les données du fichier `Dim_Date_1850-2050.csv`.

```
def populate_date_dim(cursor):
    date_dim = pd.read_csv('./Dataset/Dim_Date_1850-2050.csv')
    date_dim = date_dim.replace({np.nan: None})  # Remplace NaN par None
    for _, row in date_dim.iterrows():
        cursor.execute("""
            INSERT INTO Date_Dim (Date, Year, Month, Day)
            VALUES (%s, %s, %s, %s)
        """, (row["Date"], row["Year"], row["Month"], row["Day"]))
    print("Date_Dim populated from Dim_Date_1850-2050.csv")
```

8.2.5 `main()`

- Établit une connexion au serveur MySQL et crée la base de données.

- Se connecte à la base de données créée et initialise le schéma.
- Charge les données nettoyées et remplit les tables.

```
def main():
    initial_connection = pymysql.connect(host='localhost', user='root', password='test123',
                                         cursorclass=pymysql.cursors.DictCursor)
    create_database(initial_connection, 'weather_dataWarehouse')
    initial_connection.close()

    db = pymysql.connect(host='localhost', user='root', password='test123', database='weathe
                         charset='utf8mb4', cursorclass=pymysql.cursors.DictCursor)
    cursor = db.cursor()

    create_tables(cursor)
    print("Data Warehouse schema created")

    data = pd.read_csv('./Dataset/Weather_data.csv')
    populate_tables(cursor, data)

    # Populate DateDim separately
    populate_date_dim(cursor)

    cursor.close()
    db.commit()
    db.close()

if __name__ == "__main__":
    main()
```

8.3 Mise en Œuvre du Tableau de Bord

Le tableau de bord est mis en œuvre dans le script `Front_Dash.py`. Les principaux composants et leurs objectifs sont :

8.3.1 `fetch_data()`

- Se connecte à la base de données MySQL et récupère les données météorologiques combinées.
- Retourne les données sous forme de DataFrame pandas.

```
def fetch_data():
    db = pymysql.connect(host='localhost', user='root', password='test123', database='weathe
                         charset='utf8mb4', cursorclass=pymysql.cursors.DictCursor)
    cursor = db.cursor()
    cursor.execute("""
```

```
        SELECT *
        FROM Weather_Fact
        INNER JOIN Station_Dim ON Weather_Fact.StationID = Station_Dim.StationID
        INNER JOIN Date_Dim ON Weather_Fact.Date_ID = Date_Dim.Date_ID
    """)
    rows = cursor.fetchall()
    cursor.close()
    db.close()
    return pd.DataFrame(rows)

df = fetch_data()
```

8.3.2 Disposition de l'Application Dash

- Définit la disposition du tableau de bord, y compris les menus déroulants pour sélectionner les stations, les années, les saisons, les trimestres, les mois et les mesures.
- Inclut des conteneurs pour afficher les graphiques.

```
external_stylesheets = [
    'C:/Users/mredh/PycharmProjects/DataWarehouse2/custom.css'
]

app = dash.Dash(__name__, external_stylesheets=external_stylesheets)
server = app.server

app.layout = html.Div(
    children=[
        html.Div(
            className='header',
            children=[
                html.H1('Weather Data Dashboard')
            ]
        ),
        html.Div(
            className='container',
            children=[
                dcc.Dropdown(
                    id='station-dropdown',
                    options=[{'label': i, 'value': i} for i in df['Name'].unique()],
                    value=df['Name'].iloc[0],
                    className='dropdown'
                ),
                dcc.Dropdown(
                    id='year-dropdown',
                    options=[{'label': i, 'value': i} for i in df['Year'].unique()],
```

```
                    value=df['Year'].min(),
                    className='dropdown'
                ),
                dcc.Dropdown(
                    id='season-dropdown',
                    options=[{'label': i, 'value': i} for i in ['Spring', 'Summer', 'Autumn'
                    value='Winter',
                    className='dropdown'
                ),
                dcc.Dropdown(
                    id='quarter-dropdown',
                    options=[{'label': 'Q' + str(i), 'value': i} for i in range(1, 5)],
                    value=1,
                    className='dropdown'
                ),
                dcc.Dropdown(
                    id='month-dropdown',
                    options=[{'label': i, 'value': i} for i in df['Month'].unique()],
                    value=df['Month'].min(),
                    className='dropdown'
                ),
                dcc.Dropdown(
                    id='measure-dropdown',
                    options=[{'label': v, 'value': k} for k, v in measure_descriptions.items
                    value='TMAX',
                    className='dropdown'
                )
            ]
        ),
        html.Div(
            className='graph-container',
            children=[
                html.Div('Weather Measure Over Time', className='graph-title'),
                dcc.Graph(id='weather-graph')
            ]
        ),
        html.Div(
            className='graph-container',
            children=[
                html.Div('Distribution of Weather Measure', className='graph-title'),
                dcc.Graph(id='weather-histogram')
            ]
        )
    ]
)
```

8.3.3 Callbacks

- `update_weather_graph()` : Met à jour le graphique linéaire en fonction des filtres sélectionnés.
- `update_weather_histogram()` : Met à jour l'histogramme en fonction des filtres sélectionnés.

```
@app.callback(
    Output('weather-graph', 'figure'),
    [Input('station-dropdown', 'value'),
     Input('year-dropdown', 'value'),
     Input('season-dropdown', 'value'),
     Input('quarter-dropdown', 'value'),
     Input('month-dropdown', 'value'),
     Input('measure-dropdown', 'value')]
)
def update_weather_graph(selected_station, selected_year, selected_season, selected_quarter,
                         selected_measure):
    season_months = {
        'Spring': [3, 4, 5],
        'Summer': [6, 7, 8],
        'Autumn': [9, 10, 11],
        'Winter': [12, 1, 2]
    }

    quarter_months = {
        1: [1, 2, 3],
        2: [4, 5, 6],
        3: [7, 8, 9],
        4: [10, 11, 12]
    }

    filtered_df = df[
        (df['Name'] == selected_station) &
        (df['Year'] == selected_year) &
        (df['Month'].isin(season_months[selected_season])) &
        (df['Month'].isin(quarter_months[selected_quarter])) &
        (df['Month'] == selected_month)
        ]
    fig = px.line(filtered_df, x='Day', y=selected_measure, title=f'{measure_descriptions[se
                  template='plotly_dark')
    fig.update_traces(line=dict(color='#ff6347'))
    fig.update_layout(paper_bgcolor='white', plot_bgcolor='white')
    return fig

@app.callback(
```

```
    Output('weather-histogram', 'figure'),
    [Input('station-dropdown', 'value'),
     Input('year-dropdown', 'value'),
     Input('season-dropdown', 'value'),
     Input('quarter-dropdown', 'value'),
     Input('month-dropdown', 'value'),
     Input('measure-dropdown', 'value')]
)
def update_weather_histogram(selected_station, selected_year, selected_season, selected_quar
                             selected_measure):
    season_months = {
        'Spring': [3, 4, 5],
        'Summer': [6, 7, 8],
        'Autumn': [9, 10, 11],
        'Winter': [12, 1, 2]
    }

    quarter_months = {
        1: [1, 2, 3],
        2: [4, 5, 6],
        3: [7, 8, 9],
        4: [10, 11, 12]
    }

    filtered_df = df[
        (df['Name'] == selected_station) &
        (df['Year'] == selected_year) &
        (df['Month'].isin(season_months[selected_season])) &
        (df['Month'].isin(quarter_months[selected_quarter])) &
        (df['Month'] == selected_month)
        ]
    fig = px.histogram(filtered_df, x=selected_measure,
                       title=f'Distribution of {measure_descriptions[selected_measure]}', te
    fig.update_traces(marker_color='#ff6347')
    fig.update_layout(paper_bgcolor='white', plot_bgcolor='white')
    return fig
```

8.3.4 Fonction Principale

- Exécute le serveur de l'application Dash.

```
if __name__ == '__main__':
    app.run_server(debug=True)
```

9 Conclusion

Le projet d'entrepôt de données météorologiques transforme avec succès les données climatiques brutes en un entrepôt de données structuré en utilisant un schéma en étoile. Le processus ETL assure des données propres et cohérentes, et le tableau de bord basé sur Dash fournit des visualisations interactives et dynamiques pour analyser les données climatiques. Ce projet démontre l'efficacité de l'intégration des techniques d'entrepôt de données avec des outils de visualisation de données modernes pour une analyse complète des données climatiques.