PROJET SQL PRESENTATION

Axel Bouchaud--Roche

INTRODUCTION

Ce projet SQL offre une solution complète pour la gestion et l'analyse des données commerciales. Il permet de centraliser les informations des magasins, d'automatiser la gestion des offres, de trier et supprimer les données aberrantes et de suivre les actions grâce à un système de logs. Avec des seuils de prix dynamiques adaptés aux catégories de produits, il garantit un contrôle précis des tarifs. La génération de rapports statistiques assurent une prise de décision rapide et basée sur des données fiables.

OBJECTIFS

- Créer un processus automatisé pour l'importation des données
- Mettre à jour la table Offer
- Ajouter un mécanisme de log
- Script de vérification des données

PROCESSUS

data.sql / Aide_Import.sql

Définition de la table Store, EANItem et Offer + Insertion des données

Thresholds.sql

Définit des seuils de prix dynamiques pour des catégories de produits.

Main.sql

Le cœur du projet, avec des requêtes dynamiques pour trier et insérer les données et des logs.

Stats.sql

Génération de rapports statistiques sur les offres.

DATA.SQL : STORE

```sql
drop table if exists Store
create table Store (
    numberid bigint primary key,
    retailer int,
    description nvarchar(255),
    address nvarchar(255),
    zipcode nvarchar(10),
    city nvarchar(255),
    closingdate date,
    lat float,
    lng float,
    websitenumberid int,
    websitedescription nvarchar(255)
)
```

Création d'une table appelée Store, qui constitue la base de données principale pour stocker les informations des points de vente. Les colonnes de cette table incluent des détails essentiels tels que :

- numberid : Identifiant unique pour chaque magasin.
- retailer : Code représentant le distributeur auquel le magasin appartient.
- description : Nom ou description du magasin (par exemple, "INTERMARCHÉ HYPER SAUMUR").
- address, zipcode, et city : Informations géographiques pour localiser le magasin.
- closingdate : Date éventuelle de fermeture du magasin.
- lat et lng : Coordonnées GPS pour une localisation précise.
- websitenumberid et websitedescription : Informations relatives à la présence numérique du magasin.

DATA.SQL : EANITEM

```sql
drop table if exists EANItem
create table EANItem (
    EAN bigint,
    ItemNumberID int,
    ItemDescription nvarchar(max)
)
```

La table EANItem contient les informations des articles identifiés par leur code-barres EAN. Elle se compose de trois colonnes principales :

- EAN : Le code-barres unique de chaque produit.
- ItemNumberID : Un identifiant interne pour les articles.
- ItemDescription : Une description textuelle détaillée de chaque article.

Elle permet de référencer tous les produits disponibles, d'assurer leur traçabilité, et de faciliter les analyses ou opérations commerciales, comme la gestion des offres ou des stocks.

DATA.SQL : OFFER

```sql
drop table if exists Offer
create table Offer (
    NumberID bigint not null primary key,
    StoreNumberID bigint not null,
    StoreDescription nvarchar(255) not null,
    EAN bigint not null,
    ItemDescription nvarchar(255) null,
    PeriodStart datetime not null,
    PeriodEnd datetime not null,
    Price money null,
    VerifiedDate datetime null,
    StockShortageDate datetime null
)
```

La table Offer est utilisée pour gérer les offres commerciales liées aux produits dans les magasins. Elle contient les colonnes suivantes :

- NumberID : Identifiant unique de chaque offre (clé primaire).
- StoreNumberID : Identifiant du magasin où l'offre est disponible.
- StoreDescription : Nom ou description du magasin.
- EAN : Code-barres du produit concerné.
- ItemDescription : Description textuelle du produit.
- PeriodStart et PeriodEnd : Début et fin de la période de validité de l'offre.
- Price : Prix proposé dans le cadre de l'offre.
- VerifiedDate : Date de vérification de l'offre.
- StockShortageDate : Date de rupture de stock pour l'article.

AIDE_IMPORT.SQL : BLK_DATA

```sql
drop table if exists blk_data
create table blk_data (
    AV_EAN bigint,
    StoreNumberID int,
    LocalReco decimal(10,2),
    AppliedPrice decimal(10,2)
)
```

La table blk_data est une table temporaire utilisée pour l'importation massive de données. Elle se compose des colonnes suivantes :

- AV_EAN : Le code-barres unique des produits.
- StoreNumberID : Identifiant du magasin lié à l'article.
- LocalReco : Une valeur de recommandation locale pour l'article.
- AppliedPrice : Le prix appliqué à l'article.

Cette table sert principalement de structure intermédiaire pour recevoir les données importées depuis des fichiers externes (comme des fichiers CSV). Elle est essentielle pour préparer et valider les données avant leur intégration dans les tables principales de la base de données.

AIDE_IMPORT.SQL : BULK INSERT

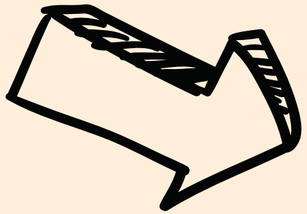
La commande BULK INSERT dans le fichier Aide Import.sql est utilisée pour importer rapidement un grand volume de données à partir d'un fichier externe (CSV) vers la table blk_data. Voici les détails :

```
declare @file nvarchar(max) = 'DATA20241111'
declare @Date date = '2024-11-02';

declare @sql nvarchar(max) = '
bulk insert blk_data
from ''C:\Users\Axel\Downloads\Projet_SQL-main\' + @file + '.csv''
with (
    fieldterminator = ''';'',
    rowterminator = ''\n'',
    firstrow = 2
)
'
exec(@sql)
```

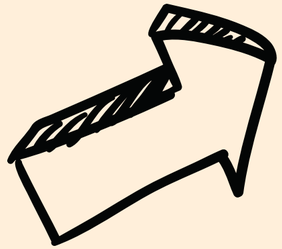
1.Source des données :

- Le fichier source est un fichier CSV dont le chemin est spécifié dynamiquement (par exemple : C:\Users\Axel\Downloads\Projet_SQL-main\DATA20241111.csv).
- Les colonnes du fichier doivent correspondre aux colonnes de la table blk_data.



2.Paramètres définis :

- fieldterminator = ';' : Définit le caractère ; comme séparateur des colonnes dans le fichier CSV.
- rowterminator = '\n' : Définit le saut de ligne comme séparateur entre les enregistrements.
- firstrow = 2 : Ignore la première ligne du fichier (souvent utilisée pour les en-têtes).



3.Exécution dynamique :

- La commande est construite dynamiquement en utilisant une variable SQL (@sql) pour s'adapter à différents fichiers.
- Elle est ensuite exécutée avec la commande EXEC(@sql).

THRESHOLDS.SQL : CATEGORY_THRESHOLD

```sql
-- Step 1: Create CategoryThreshold Table (if it does not exist)
IF OBJECT_ID('CategoryThreshold', 'U') IS NULL
CREATE TABLE CategoryThreshold (
    Category VARCHAR(50) PRIMARY KEY,
    MinPrice DECIMAL(10, 2),
    MaxPrice DECIMAL(10, 2)
);
```

La table CategoryThreshold définit des seuils de prix pour différentes catégories de produits. Elle contient :

- Category : Le nom de la catégorie.
- MinPrice et MaxPrice : Les prix minimum et maximum autorisés pour cette catégorie.

Elle est utilisée pour vérifier que les produits respectent les limites de prix définies, garantissant une gestion cohérente des politiques tarifaires.

THRESHOLDS.SQL : INSERTION DES CATÉGORIES

```sql
-- Step 2: Extract Categories from ItemDescription and Insert Defaults
INSERT INTO CategoryThreshold (Category, MinPrice, MaxPrice)
SELECT DISTINCT
    TRIM(LEFT(ItemDescription, CHARINDEX(')', ItemDescription))) AS Category,
    1 AS MinPrice, -- Default MinPrice
    1000 AS MaxPrice -- Default MaxPrice
FROM Offer
WHERE NOT EXISTS (
    SELECT 1
    FROM CategoryThreshold CT
    WHERE CT.Category = TRIM(LEFT(Offer.ItemDescription, CHARINDEX(')', Offer.ItemDescription)))
);
```

Dans le fichier Thresholds.sql, les catégories de produits sont ajoutées automatiquement à la table CategoryThreshold.

- Les noms des catégories sont extraits des descriptions d'articles.
- Chaque nouvelle catégorie détectée est insérée avec des seuils par défaut, par exemple un prix minimum de 1 et un maximum de 1000.

Cela garantit que toutes les catégories sont correctement enregistrées et prêtes à être utilisées pour le contrôle des prix.

THRESHOLDS.SQL : UPDATE DES SEUILS

```sql
-- Update Thresholds for Existing Categories in CategoryThreshold Table

-- J'ai pas eu le courage de tout optimiser, bonne chance...

-- Alimentaire
UPDATE CategoryThreshold SET MinPrice = 5, MaxPrice = 500 WHERE Category = '(Alimentaire)';

-- Artisanat
UPDATE CategoryThreshold SET MinPrice = 5, MaxPrice = 1000 WHERE Category = '(Artisanat)';

-- Audio
UPDATE CategoryThreshold SET MinPrice = 5, MaxPrice = 800 WHERE Category = '(Audio)';

-- Automobile
UPDATE CategoryThreshold SET MinPrice = 5, MaxPrice = 500 WHERE Category = '(Automobile)';

-- Beauté
UPDATE CategoryThreshold SET MinPrice = 5, MaxPrice = 1000 WHERE Category = '(Beauté)';

-- Bureau
UPDATE CategoryThreshold SET MinPrice = 5, MaxPrice = 500 WHERE Category = '(Bureau)';

-- Cuisine
UPDATE CategoryThreshold SET MinPrice = 5, MaxPrice = 1000 WHERE Category = '(Cuisine)';
```

CECI N'EST QU'UN EXTRAIT DE TOUTES LES UPDATES

Dans le fichier Thresholds.sql, la mise à jour des seuils de prix dans la table CategoryThreshold est réalisée pour ajuster les valeurs existantes en fonction de nouvelles conditions ou données.

- Mise à jour ciblée : Les seuils MinPrice et MaxPrice sont modifiés pour des catégories spécifiques selon des critères définis (par exemple, basés sur les données des articles ou des analyses de marché).

MAIN.SQL : GESTION DES LOGS

```sql
-- Log Inserted Offers
DROP TABLE IF EXISTS Log
IF OBJECT_ID('Log', 'U') IS NULL
CREATE TABLE Log (
    LogID INT IDENTITY(1,1) PRIMARY KEY,
    LogDate DATETIME DEFAULT GETDATE(),
    Action VARCHAR(50),
    Details NVARCHAR(MAX)
);
```

Création d'une table Log (si elle n'existe pas déjà) pour suivre les actions réalisées par les scripts.

Colonnes importantes dans la table :

- LogID : Identifiant unique pour chaque action.
- LogDate : Date de l'action.
- Action : Description de l'action (par exemple, insertion, mise à jour).
- Details : Détails supplémentaires sur l'action.

Cette table permet de tracer toutes les modifications ou les actions importantes pour assurer une meilleure transparence.

MAIN.SQL : LOGS DES NOUVELLES OFFRES

```sql
-- Log Newly Created Offers
INSERT INTO Log (Action, Details)
SELECT
    'Insert',
    CONCAT('Created new offer for EAN ', B.AV_EAN,
           ' in store ', B.StoreNumberID,
           ' with start date ', @FileDate,
           ' and end date 9999-12-31')
FROM blk_data B
LEFT JOIN Offer O ON O.EAN = B.AV_EAN AND O.StoreNumberID = B.StoreNumberID
WHERE O.EAN IS NULL; -- Log only new offers
```

Ce script garantit que chaque nouvelle offre insérée dans la base est documentée dans la table Log, avec des informations détaillées sur son origine et sa période de validité.

On vérifie si l'EAN du produit n'est pas déjà présent.

MAIN.SQL : NOUVELLES OFFRES

```sql
-- Insert Offers (Active and Inactive) with PeriodEnd = '9999-12-31'
INSERT INTO Offer (
    NumberID, StoreNumberID, StoreDescription, EAN, ItemDescription, Price, VerifiedDate, PeriodStart, PeriodEnd
)
SELECT
    ISNULL((SELECT MAX(NumberID) FROM Offer), 0) + ROW_NUMBER() OVER (ORDER BY B.AV_EAN) AS NumberID,
    B.StoreNumberID,
    S.Description AS StoreDescription,
    B.AV_EAN,
    ISNULL(E.ItemDescription, 'Unknown Item') AS ItemDescription, -- Default for missing EANs
    B.AppliedPrice,
    GETDATE(), -- Verified date
    @FileDate, -- PeriodStart
    '9999-12-31' -- Default PeriodEnd for all offers
FROM blk_data B
LEFT JOIN Store S ON B.StoreNumberID = S.NumberID
LEFT JOIN EANItem E ON B.AV_EAN = E.EAN
LEFT JOIN Offer O ON O.EAN = B.AV_EAN AND O.StoreNumberID = B.StoreNumberID
WHERE O.EAN IS NULL; -- Only new offers

INSERT INTO Log (Action, Details)
SELECT
    'Insert',
    CONCAT('Inserted offer for EAN ', B.AV_EAN, ' in store ', B.StoreNumberID)
FROM blk_data B
LEFT JOIN EANItem E ON B.AV_EAN = E.EAN
LEFT JOIN Offer O ON O.EAN = B.AV_EAN AND O.StoreNumberID = B.StoreNumberID
WHERE O.EAN IS NULL;
```

Ce script :

1. Insère de nouvelles offres en s'assurant qu'elles ne sont pas déjà présentes.
2. Génère dynamiquement les IDs et ajoute des valeurs par défaut pour certains champs.
3. Enregistre chaque insertion dans une table de Log pour assurer une traçabilité complète.

Cela garantit une gestion robuste et cohérente des données d'offres dans la base de données.

MAIN.SQL :
LOG DES UPDATES DES OFFRES

```sql
-- Log Updated Offers Before Changing Data
INSERT INTO Log (Action, Details)
SELECT 'Update',
       CONCAT('Updated offer for EAN ', B.AV_EAN,
              ' in store ', B.StoreNumberID,
              ' from price ', O.Price,
              ' to price ', B.AppliedPrice)
FROM Offer O
JOIN blk_data B
    ON O.EAN = B.AV_EAN AND O.StoreNumberID = B.StoreNumberID
WHERE O.Price <> B.AppliedPrice; -- Capture differences before updating
```

Ce script insère des enregistrements dans la table Log pour tracer les mises à jour des offres dans la table Offer, avant que les modifications ne soient effectuées.

- Vérifie que le prix actuel (O.Price) est différent du nouveau prix proposé (B.AppliedPrice).
- Seules les offres où une modification de prix est détectée sont ajouter dans Log.

MAIN.SQL : UPDATE DES OFFRES

```sql
-- Update Existing Offers (Price and VerifiedDate)
UPDATE O
SET O.Price = B.AppliedPrice,
    O.VerifiedDate = GETDATE()
FROM Offer O
JOIN blk_data B
    ON O.EAN = B.AV_EAN AND O.StoreNumberID = B.StoreNumberID
WHERE O.Price <> B.AppliedPrice; -- Actual update
```

Ce script effectue une mise à jour des offres déjà présentes dans la table Offer en ajustant leur prix et leur date de vérification si des modifications sont nécessaires.

- La mise à jour est effectuée uniquement si le prix actuel (O.Price) est différent du nouveau prix proposé (B.AppliedPrice).
- Cela empêche les mises à jour inutiles pour les lignes où les prix sont déjà corrects.

MAIN.SQL : FERMETURE DES OFFRES

```sql
-- Close Offers Not Observed in blk_data after update
IF EXISTS (SELECT 1 FROM blk_data)
    UPDATE O
    SET O.PeriodEnd = @FileDate
    FROM Offer O
    LEFT JOIN blk_data B ON O.EAN = B.AV_EAN AND O.StoreNumberID = B.StoreNumberID
    WHERE B.AV_EAN IS NULL AND O.PeriodEnd = '9999-12-31';
```

Ce script met à jour les offres dans la table Offer pour marquer comme expirées celles qui ne sont plus présentes dans les nouvelles données importées (blk_data).

- B.AV_EAN IS NULL : Sélectionne les offres qui ne figurent pas dans blk_data, c'est-à-dire celles non importées.
- O.PeriodEnd = '9999-12-31' : Limite l'action aux offres qui sont encore actives (celles dont la période de fin est définie à la valeur par défaut "9999-12-31").

MAIN.SQL : LOG DES FERMETURES DES OFFRES

```sql
-- Log Closed Offers
INSERT INTO Log (Action, Details)
SELECT 'Close',
       CONCAT('Closed offer for EAN ', O.EAN, ' in store ', O.StoreNumberID, ' on ', @FileDate)
FROM Offer O
LEFT JOIN blk_data B ON O.EAN = B.AV_EAN AND O.StoreNumberID = B.StoreNumberID
WHERE B.AV_EAN IS NULL AND O.PeriodEnd = @FileDate;
```

Ce script insère des enregistrements dans la table Log pour documenter les offres marquées comme fermées dans la table Offer après leur mise à jour.

- B.AV_EAN IS NULL : Sélectionne les offres qui ne figurent pas dans blk_data (offres absentes dans les nouvelles données).
- O.PeriodEnd = @FileDate : Limite l'action aux offres qui ont été récemment fermées (lors de la mise à jour précédente).

MAIN.SQL : LOG DES PRIX ABERRANTS

```sql
-- Log Anomalous Prices Based on Dynamic Thresholds
INSERT INTO Log (Action, Details)
SELECT
    'Anomaly',
    CONCAT('Anomalous price detected: ', B.AppliedPrice,
           ' for EAN ', B.AV_EAN,
           ' in category ', LEFT(O.ItemDescription, CHARINDEX(')', O.ItemDescription)))
FROM blk_data B
LEFT JOIN Offer O ON B.AV_EAN = O.EAN
LEFT JOIN CategoryThreshold C
    ON LEFT(O.ItemDescription, CHARINDEX(')', O.ItemDescription)) = C.Category
WHERE B.AppliedPrice < C.MinPrice OR B.AppliedPrice > C.MaxPrice;
```

Ce script enregistre dans la table Log les produits dont le prix est détecté comme anormal, basé sur les seuils dynamiques définis dans la table CategoryThreshold.

- Vérifie si le prix appliqué (B.AppliedPrice) est :
  - Inférieur au prix minimum défini (C.MinPrice).
  - Supérieur au prix maximum défini (C.MaxPrice).

MAIN.SQL : SUPPRESSION DES PRIX ABERRANTS

```sql
-- Remove Anomalous Prices
DELETE O
FROM Offer O
    INNER JOIN CategoryThreshold C
    ON LEFT(O.ItemDescription, CHARINDEX(')', O.ItemDescription)) = C.Category
WHERE O.Price < C.MinPrice OR O.Price > C.MaxPrice;
```

Ce script supprime de la table Offer toutes les offres dont les prix sont considérés comme anormaux, c'est-à-dire situés en dehors des seuils définis dans la table CategoryThreshold.

- O.Price < C.MinPrice : Identifie les offres avec un prix inférieur au minimum défini pour la catégorie.
- O.Price > C.MaxPrice : Identifie les offres avec un prix supérieur au maximum défini pour la catégorie.

STATS.SQL

Stats.sql génère un rapport statistique détaillé sur les offres, regroupées par magasin, pour fournir des informations clés sur les performances des offres.

```sql
-- Rapport statistique sur la table Offre
SELECT
    S.NumberID AS StoreID,
    S.Description AS StoreName,
    COUNT(O.NumberID) AS TotalOffers,
    SUM(CASE WHEN O.PeriodEnd = '9999-12-31' THEN 1 ELSE 0 END) AS ActiveOffers,
    SUM(CASE WHEN O.PeriodEnd < GETDATE() THEN 1 ELSE 0 END) AS ClosedOffers,
    SUM(CASE WHEN O.StockShortageDate IS NOT NULL THEN 1 ELSE 0 END) AS StockShortages,
    AVG(O.Price) AS AvgPrice,
    MIN(O.Price) AS MinPrice,
    MAX(O.Price) AS MaxPrice
FROM Offer AS O
LEFT JOIN Store AS S
ON O.StoreNumberID = S.NumberID
GROUP BY S.NumberID, S.Description
ORDER BY StoreID;
```

TotalOffers :

Le nombre total d'offres pour chaque magasin, calculé avec COUNT(O.NumberID).

ActiveOffers :

Nombre d'offres encore actives, définies par la condition O.PeriodEnd = '9999-12-31'.

StockShortages :

Nombre d'offres marquées comme ayant une rupture de stock, détectées par O.StockShortageDate IS NOT NULL.

AvgPrice, MinPrice, et MaxPrice :

Calcul respectivement de la moyenne, du minimum, et du maximum des prix des offres (O.Price).

ClosedOffers :

Nombre d'offres expirées, avec une date de fin (PeriodEnd) antérieure à la date actuelle (GETDATE()).

CONCLUSION

Ce projet SQL propose une gestion complète et automatisée des offres commerciales, des seuils de prix, et des données magasins. Les scripts permettent de :

- Créer et importer des données en masse dans des structures bien définies, comme les tables des magasins, des articles et des offres.
- Identifier, insérer, mettre à jour ou supprimer des offres en fonction des données reçues, tout en assurant une traçabilité grâce à une table de journalisation.
- Contrôler les prix dynamiquement via des seuils pour détecter et traiter les anomalies tarifaires, en supprimant les données incohérentes.
- Générer des rapports statistiques détaillés par magasin, incluant des métriques sur les offres actives, expirées, en rupture de stock, ainsi que sur les prix moyens, minimaux et maximaux.

Grâce à ces scripts, le système garantit une base de données cohérente, actualisée et exploitable pour les analyses et la prise de décision.

MERCI