Projet de ETL

Contexte :

La direction d'AdventureWorks (AW) chante vos louanges depuis maintenant plusieurs semaines. Chaque fois que vous leur apportez une solution à un problème d'affaires qu'ils rencontrent, vous allez au-delà de leur besoin et trouvez une solution innovante et facile d'emploi. Bref, il se pourrait que vous soyez bientôt employé.e du mois chez AW! C'est important parce que cette reconnaissance est accompagnée d'une récompense sous la forme d'une carte-cadeau d'une valeur de $10 (wow!).

AW aimerait explorer la possibilité de créer une petite structure de données pour une preuve de concept pour créer une sorte de « cube » de données pour les ventes. L'idée vous est présentée de la façon suivante :

- « Ce qu'on voudrait c'est une structure très simple, avec une dimension de date, ainsi qu'une table de fait pour les ventes. On pourrait les appeler 'dim_date' et 'fact_sales_cube'.
- Dim_date est une dimension de date bien classique comportant les attributs suivants : un identifiant unique pour la date, la valeur de la date, la valeur du mois (numérique), la valeur du jour de la semaine (numérique), la valeur de l'année (numérique), et la valeur de l'année fiscale.
- Chez AW, l'année fiscale commence le premier lundi du mois d'avril de chaque année. Donc en 2022 par exemple, l'année fiscale 2022 a commencé le 4 avril. Avant ça on était encore en 2021.
- La dimension de date peut être générée avec les données du 1er janvier 2010 au 1er janvier 2015 au moment où on crée la table.
- Fact_sales_cube est une table de fait qui va contenir une espèce de produit croisé des dates et des produits. Il faut le voir de la façon suivante : pour une date donnée (liée à dim_date avec l'identifiant de la date), on devrait avoir une ligne avec les ventes de chaque produit. Autrement dit, s'il y a 10 produits dans la base de données et 30 jours dans un mois, on aura 300 lignes pour ce mois, même si on n'a pas vendu chacun des 10 produits pendant chaque jour de ce mois-là. Et ainsi de suite pour chaque mois qu'on voudrait générer dans la table.
- L'alimentation de cette table se fera à l'aide d'un objet SQL qu'on pourra appeler par exemple via un « job » automatique pour générer les données des ventes pour un mois qui vient de se clore. Il faudra donc que l'objet génère les données en fonction d'une combinaison année/mois qu'on lui aura donnée, et on voudra aussi avoir la possibilité de lui dire non pas de remplir les informations pour cette combinaison année/mois, mais aussi de les effacer parce que si on génère les données pour un mois donné mais qu'on a des erreurs à corriger dans le système transactionnel d'AW, il faut que ce soit facile de regénérer les données pour ce mois (on appellerait l'objet une première fois pour « flusher » les données de cette année/mois, et une deuxième fois pour les générer). En même temps, si quelqu'une tente de générer les données pour un mois déjà traité (sans indiquer qu'il/elle veut flusher les données), on voudra arrêter le processus et afficher un beau message d'erreur comme quoi ce n'est pas possible de le faire, vu qu'on a déjà des données pour ce mois de l'année.

- Il va aussi falloir que tout ça soit clean parce que l'architecte en chef va repasser dans ton code après et il a une certaine influence sur le CSSCCPCAW (le Comité de Sélection de la Super Carte Cadeau Pas Cheap d'AdventureWorks).
    - Un truc cool, ce serait que le chargement de la table de fait fonctionne avec des transactions, comme ça on sait que le chargement passe au complet ou ne passe pas du tout, pour éviter les problèmes d'intégrité des données. »

« Facile comme commande, les $10 sont déjà à moi! » vous vous dites, une fois retourné.e à votre bureau. Bon évidemment, l'architecte a quand même laissé des consignes techniques à respecter, que voici :

- « Vous devez créer un seul fichier (script) SQL, qui contiendra des sections bien identifiées avec des commentaires permettant de relire votre code et de comprendre ce qu'il fait.
- Vous devez également avoir un entête dans votre script (voir les hauts de scripts des autres TPs pour vous inspirer).
- Je dois pouvoir prendre le script, et l'exécuter dans son entièreté pour que tout soit systématiquement créé et effacé au besoin. Autrement dit, je ne veux pas avoir à sélectionner des morceaux du script pour tout faire ou à effacer des trucs moi-même pour pouvoir recréer les objets. Ça implique donc d'effacer conditionnellement les objets avant de les créer, de mettre des GO là où c'est nécessaire etc.
- La première partie du script doit effacer/créer une base de données appelée 'tp_etl'.
- La deuxième partie du script doit créer deux tables :
    - Dim_date (cf. plus haut pour les attributs à inclure)
    - Fact_sales_cube avec les attributs suivants (n'oubliez pas de valider que vous utilisez les bons types de données):
        - ProductNumber (numéro du produit).
        - SubCategory (produit).
        - Category (produit).
        - Date_id (identifiant de la date située dans la dimension dim_date).
        - TotalQuantity : quantité totale commandée pour ce produit dans une journée donnée.
        - TotalAmount : montant totale vendu pour ce produit dans une journée donnée.
        - OrderCount : nombre de commandes pour ce produit dans une journée donnée.
- La troisième partie doit remplir la dimension de date entre le 1er janvier 2010 et le 1er janvier 2015
    - Pas besoin de procédure pour cela, on peut simplement faire un INSERT dans le script principal.
    - Conseil : avant de tenter de calculer l'année fiscale (ou si vous bloquez là-dessus), remplissez la dimension en mettant une valeur arbitraire, et faites le reste du devoir. Vous pourrez toujours revenir sur cette partie des requis plus tard.
- La quatrième partie doit créer l'objet pour charger fact_sales_cube :

- L'objet doit uniquement insérer les données pour chaque jour de la combinaison année/mois passée en paramètre. Attention, on veut du code clean et on veut des transactions avec des COMMIT/ROLLBACK en cas de problème!
- Si on a déjà des données pour une combinaison mois/année passée en paramètre, il faut tout arrêter et soulever une erreur à l'utilisateur en indiquant un message comme quoi la combinaison année/mois passée en paramètre est déjà chargée. Le message doit afficher le mois/année qui avait été donné.
- Pour la catégorie et la sous-catégorie d'un produit, on veut avoir la valeur « N/A » plutôt que des valeurs NULL.
- L'objet doit aussi accepter un troisième paramètre, @aFlushData, de type bit (i.e., *boolean*). Par défaut, la valeur de ce paramètre est 0, mais si on passe 1, alors, on efface les données de fact_sales_cube pour les jours de cette combinaison mois/année. L'exécution de cette action doit être accompagnée d'un message d'information indiquant qu'on a effacé les données avec le mois/année (par exemple « Data deleted for 2013/04 »).
- Quand on charge les données dans fact_sales_cube, on va donc avoir une ligne par jour par produit. Si on n'a rien vendu pour ce produit dans cette journée, on aura tout simplement besoin d'afficher 0 pour les mesures quantitatives de cette table. »

- **Comment tester votre code :**
  - Votre script doit inclure les tests suivants (en prenant pour hypothèse que votre objet et vos tables viennent tout juste d'être créés):
    - 1 premier appel à votre objet pour mars 2012 qui doit fonctionner
    - 1 deuxième appel à votre objet pour mars 2012 qui doit retourner une erreur comme quoi les données pour mars 2012 existent déjà
    - 1 appel à votre objet pour mars 2012 indiquant qu'on veut effacer les données et qui doit fonctionner en affichant un message indiquant que les données ont été effacées pour mars 2012
    - 1 dernier appel à votre objet pour mars 2012 qui doit fonctionner étant donné que les données avaient été effacées à l'appel précédent.
  - d'autres objets au besoin si cela vous simplifie la vie!