EXERCICES FORMATION SQL niveau 2

04/01/2024 - 11/01/2024

Tuteur : **Guillaume Vayeur**

I - Analyse d’énoncé

Ce sujet est composé par deux parties, d’une première partie pour initialiser les tables et les données , et une deuxième partie où on manipule les données.
La première partie je vais appeler partie I, et la deuxième partie je vais appeler partie II dans la suite.

Tout d’abord, on commence par initialiser les tables et les données dans la partie I à l’aide des quatre premières questions. La pièges de cette partie est :

- short_name n’est pas clair, faut confirmer avec le tuteur . J’ai pris celui qui se trouve sur WIKIPEDIA(**CODE-ISO 31662**).
- Faut faire attention, car dans la table departement, il y a departement_id(INT) et departement_code(VARCHAR) , j’ai pris le departement_id initialement, mais ensuite j’ai changé à departement_code pour faciliter les tâches.

La partie II commence à partir de la question 5, pour laquelle tous les codes qu’on fait serait dans un procédure stocké. Il y a plusieurs pièges dans cette partie à éviter et à retenir pour plus tard :

- Quand une signature d’une procédure est donnée, on doit la respecter.
- On doit lire l’intégralité du sujet, et décider quelle question à faire en première.
    - Ici, on doit **commencer d’abord par l’implémentation des flux JSON** pour filtrer les données dans la table
    - Puis on implémente les codes pour les deux modes (mode ‘C’ et mode ‘R’) ,
    - Les contraintes sont à faire à la fin, car c’est quand les codes marchent correctement, on ajoute des contraintes.

Tableau de correspondance **CODE-ISO 31662**:

Nom région	CODE-ISO	Nom région	CODE-ISO
Auvergne-Rhône-Alpes	ARA	Nouvelle-Aquitaine	NAQ
Bourgogne-Franche-Comté	BFC	Occitanie	OCC
Bretagne	BRE	Pays de la Loire	PDL
Centre-Val de Loire	CVL	Provence-Alpes-Côte d"Azur	PACA
Corse	COR	Guadeloupe	GLP
Grand Est	GE	Guyane	GUF
Hauts-de-France	HDF	Martinique	MTQ
Île-de-France	IDF	La Réunion	REU
Normandie	NOR	Mayotte	MYT

II - Amélioration éventuelles

1. Mettre en place un curseur pour la lecture de JSON.
    - Ce que j'ai fait : Stocker les données JSON dans une table, puis grâce à une select pour construire la condition While :

```
declare @where varchar(max) = ''
IF exists (SELECT 1 FROM #tmp_critere WHERE type_critere <> 'multiple')
BEGIN
    SELECT @where = @where +
    CASE
        WHEN @where <> '' THEN ' AND ' ELSE ''
    END
    + CASE                    .
        WHEN type_critere = 'like' THEN
            nom_critere + ' LIKE '''    + valeur_critere + ''' '
        WHEN type_critere = 'compare|inf' THEN
            nom_critere + ' < '     + valeur_critere
        WHEN type_critere = 'compare|sup' THEN
            nom_critere + ' > '     + valeur_critere
        ELSE
            ''
    END
    FROM #tmp_critere  WHERE type_critere <> 'multiple'
END
```

    - Comme chaque ligne de flux JSON doit traîter individuellement, c'est peut être mieux d'utiliser un curseur ici.
2. Séparer les conditions, et les appliquer sur des tables différentes.
    - Ce que j'ai fait : Je construis une table qui est la jointure des quatre tables (3 tables des données, +1 pour surface département), puis j'applique tous les conditions sur cette table:

```
declare @query varchar(max) =
'   INSERT INTO #tmp_ville(ville_id, short_name, departement_code )
    SELECT v.ville_id, r.short_name, d.departement_code
    FROM villes_france_free v
    LEFT JOIN departement_cyu d ON v.ville_departement = d.departement_code
    LEFT JOIN region_cyu r ON d.region_id = r.region_id
    JOIN departement_tmp dt ON d.departement_nom = dt.departement_nom
    WHERE
'
```

    Et puis stocker chaque colonne dans une table correspondant pour filtrer les données
    - Ce que je pourrais améliorer: Séparer les conditions en @where_ville, @where_region et @where_departement.

3. Jointure au lieu de "IN" pour le critère 'multiple' .
    Ce que j'ai fait : J'utilise 'IN' pour les critères 'multiple'

Ce que je pourrais améliorer: Utiliser la ‘JOINTURE’ pour traîter les critères ‘multiple’

III - Commentaire du code

1. Utilisation du curseur pour la table 3

La table Ville_france_free contient 36700 lignes, si le filtrage n’est pas assez fort et qu’il reste encore plus de 8000 lignes, alors un erreur va se produire lorsqu’on alimente la table @tmpTable pour l’affichage :

String or binary data would be truncated

Donc je décide d’appliquer le curseur ici pour éviter ce problème.

2. Utilisation du curseur pour @tmp_critere
@tmp_critere est la table qui contient toutes les colonnes du flux JSON.

```
BEGIN
    if dbo.F_cyu_translation(@name) <> ''
    BEGIN
        insert into #tmp_critere (nom_critere, type_critere, valeur_critere)
        VALUES (dbo.F_cyu_translation(@name), @type_critere, @valeur_critere)
        FETCH NEXT FROM critere_curseur INTO @name, @type_critere, @valeur_critere
    END
    else
    BEGIN
        SELECT  @codeError = -1, @messageError = 'Unknown column name: ' + @name
        BREAK;
    END
END
```

Dans le curseur, j'insère les lignes un par un, et quand il y a un nom de colonne inconnu, on retourne un message erreur.

3. Explication sur la filtrage des données

```
declare @query varchar(max) =
'   INSERT INTO #tmp_ville(ville_id, short_name, departement_code )
    SELECT v.ville_id, r.short_name, d.departement_code
    FROM villes_france_free v
    LEFT JOIN departement_cyu d ON v.ville_departement = d.departement_code
    LEFT JOIN region_cyu r ON d.region_id = r.region_id
    JOIN departement_tmp dt ON d.departement_nom = dt.departement_nom
    WHERE
'
```

Mon idée est de faire une jointure , pour laquelle j’aurai tous les colonnes de la table ville, département , région, et departement_tmp(qui contient la surface département)
Et puis appliquer tous les conditions while sur cette table .

La table de résultat contient trois colonnes, qui sont les villes qui vérifient les conditions introduit par le flux JSON:

- ville_id
- short_name
- departement_code

Enfin je crée trois tables temporaire pour stocker chaqu'une de ces colonnes

```
select ville_id
INTO #ville_filter
FROM #tmp_ville

select DISTINCT short_name
INTO #region_filter
FROM #tmp_ville

select DISTINCT departement_code
INTO #departement_filter
FROM #tmp_ville
END
```

IV - Récapitulation technique

1. Utilisation de curseur
   - 1.1. Quand on doit traîter beaucoup des données, et que le système n'est pas capable de traîter autant des données d'un seul coup
   - 1.2. Quand des lignes peuvent causer des fin d'exécution(Comme dans le cas de JSON)
2. Procédure et fonction
   - 2.1. Procédure a pour but principal d'exécuter un morceau de code, tandis que la fonction a pour but de retourner une valeur.
   - 2.2. Une fonction peut être appelée partout(Dans une select, dans une procédure), tandis qu'un processus doit être appelé à part.
   - 2.3. On ne peut pas utiliser GO dans une procédure
   - 2.4. On ne peut pas appeler une table temporaire dans une fonction
3. GO dans SQL Server
   - 3.1. Permet d'isoler le contexte de chaque portion de code et leurs variables. La portée d'une variable n'est alors valable que depuis le début du code ou la fin du GO à un autre GO.
   - 3.2. Permet de pouvoir créer des objets dans un seul fichier. (CREATE TABLE , CREATE PROCEDURE, DROP TABLE , etc.)
   - 3.3. Permet de répéter l'exécution de code N fois.
4. Utilisation de paramètres OUTPUT d'un procédure
   - 4.1. L'avantage est qu'il peut retourner autant de valeur par l'utilisation des paramètres OUTPUT.
5. Utilisation de flux JSON (ISJSON, OPENJSON, etc.)

5.1. Quand on reçoit une requête JSON, on peut utiliser 'ISJSON' pour vérifier s'il contient une erreur dans la requête.
6. Utilisation des fonctions en SQL
7. Utilisation des requêtes dynamiques
8. Ne pas insérer des ID en dur, mais par une select
   8.1. Car si une ligne s'est fait supprimer, et s'est ajoutée à nouveau, son ID change dans ce cas.
9. STRING_AGG, CONVERT, ISNULL,
10. Vérification si une table temporaire existe
    10.1. IF OBJECT_ID('tempdb..#ville_filter) IS NULL
          return -1
    - tempdb est un constant, on l'ajoute toujours
    - .. : est une abréviation de '.dbo.'
    - #ville_filter est le nom de la table temporaire .
11. Tips : SELECT COUNT(1) FROM ...
    11.1. Quand on a pas de jointure, et qu'on veut simplement savoir combien de lignes y a dans une table, on peut utiliser simplement 'SELECT COUNT(1) FROM ...', ici le (1) indique la première colonne.
12. Dans une boucle WHILE pour JSON, on peut utiliser BREAK pour sortir.
    12.1. Dans l'exercice de la formation, on utilise WHILE pour construire une requête . On appelle BREAK quand on trouve un nom de la colonne inconnue , et puis on retourne un code erreur .
13. exec (@query) , pour exécuter une requête stockée dans une variable.
14. On a pas besoin de supprimer une table temporaire explicitement, il serait supprimé automatiquement après la fin d'exécution.