TP5 : Données semi-structurées avec XML

Objectif général

Dans ce TP, le but est de mettre en œuvre le type XML dans PostgreSQL pour enrichir la base *entreprises* avec des informations variées, hiérarchiques, impossibles ou lourdes à modéliser avec des colonnes fixes.

Contexte

Votre entreprise suit les employés, projets et salaires dans des tables relationnelles classiques. Cependant, certains besoins métiers dépassent la rigidité du schéma classique. On veut stocker :

- Des évaluations détaillées d'employés (compétences, feedbacks, bonus personnalisés, etc.)
- Des rapports de missions ou livrables rattachés à un projet (contenus riches, structurés différemment selon le projet)

Ces données sont semi-structurées, variables, et souvent hiérarchiques, il faut donc les représenter et les manipuler avec du XML.

Structure existante

Actuellement, vous disposez des tables suivantes dans votre BD :

- employes(id, nom, prenom poste, salaire, date_embauche, id_departement , prime)
- projets(id, nom_projet, date_début, date_fin, budget)
- employe_projet(emp_id, projet_id)
- departements(id, nom_departement)
- audit_salaire(id, employe_id, ancien_salaire, nouveau_salaire, action_type, date_modification, utilisateur)

Exercice 1 : Intégration des évaluations et compétences

Problème :

Chaque employé a une fiche d'évaluation annuelle contenant des éléments variables :

- notes sur plusieurs compétences (technique, communication…)
- commentaires libres du manager
- bonus spécifiques
- remarques RH éventuelles

Dans ce contexte, nous n'avons pas de structure fixe = inadapté à un modèle tabulaire.

ToDo :

Question 1 :

Créer une table *fiche_evaluation* permettant de stocker les évaluations annuelles des employés. Elle doit contenir les champs suivants (Structure hybride):

- id : identifiant unique
- id_employe : identifiant de l'employé (clé étrangère)
- annee : année de l'évaluation
- details : champ de type XML contenant les informations détaillées

Syntaxe à utiliser pour créer un champ XML dans PostgreSQL :

```
Nom_champ XML
```

Question2 :

- Insérer au moins 5 fiches d'évaluation (avec des données variées) en respectant la structure XML (XML Schema) ci-dessous.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
       elementFormDefault="qualified">

 <xs:element name="evaluation">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="competences">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="technique" type="xs:positiveInteger"/>
       <xs:element name="communication" type="xs:positiveInteger"/>
       <xs:element name="initiative" type="xs:positiveInteger"/>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name="commentaire" type="xs:string"/>
    <xs:element name="bonus" minOccurs="0">
     <xs:complexType>
      <xs:attribute name="montant" type="xs:decimal" use="required"/>
```

```
        <xs:attribute name="raison" type="xs:string" use="required"/>
       </xs:complexType>
     </xs:element>
    </xs:sequence>
   </xs:complexType>
 </xs:element>

</xs:schema>
```

- Quelques explications :
  - **positiveInteger** : valeurs strictement positives pour les notes
  - **bonus est optionnel (minOccurs="0")** car il se peut que certains n'en aient pas
  - Les attributs montant et raison sont requis si <bonus> est présent

- Syntaxe pour insérer du XML dans PostgreSQL

```
INSERT INTO nom_table (colonne_xml) VALUES (
$$
<balise> ... </balise>
$$
);
```

Question3 :

Remarque importante :

PostgreSQL ne supporte pas le langage XQuery (vu en cours), mais supporte le langage XPath 1.0, que l'on peut utiliser via la fonction xpath(). Via ce TP, vous allez manipuler XPath pour l'interrogation des données XML. Des exemples de syntaxe sont indiqués dans l'annexe **(Voir Annexe1)** à la fin de ce TP.

Utilisez la fonction xpath() de PostgreSQL pour répondre aux questions suivantes :

- Extraire les notes de la compétence communication pour tous les employés.
- Lister les employés qui ont obtenu un bonus supérieur à 1000€.
- Extraire tous les commentaires présents dans les fiches d'évaluation.
- Afficher les employés ayant une note en initiative égale à 5.
- Extraire les notes technique et communication de chaque fiche, sur une même ligne.
- Lister les fiches avec un commentaire incluant un mot-clé, mais uniquement si la note technique est supérieure à 3 et le bonus est supérieur à 400€.

- Extraire les fiches des employés ayant une note en communication inférieure à 3, un bonus supérieur à 1000€, et un commentaire mentionnant un projet spécifique.

Question Bonus :

Vous êtes chargé de fusionner les fiches d'évaluation de l'employé id_employe = 1 (ou un autre employé de votre choix) en un seul document XML. Pour chaque employé, vous devez combiner les informations suivantes de ses fiches d'évaluation :

- Tous les commentaires présents dans les différentes fiches.
- Le bonus total de l'employé (en additionnant les bonus de toutes ses fiches d'évaluation).

Vous utiliserez une table virtuelle (CTE – Common Table Expression) pour agréger les données nécessaires à la création d'un nouveau document XML combiné. **(Voir Annexe3)**

Directives :

- Création d'une table virtuelle (CTE) : Utilisez une CTE pour sélectionner les fiches d'évaluation de l'employé spécifié. Cette CTE devra regrouper les commentaires et calculer le bonus total de l'employé à partir des données XML contenues dans le champ details.
- Dans la CTE, agrégerez les commentaires en une seule chaîne de texte à l'aide de la fonction **string_agg()** et calculez la somme des bonus avec **SUM(). (Voir Annexe2)**
- Dans la requête principale, vous utiliserez **xmlconcat()** et **xmlparse()** pour générer un nouveau document XML qui contient :
  - Le commentaire global fusionné.
  - Le bonus total calculé.
- Le document XML doit être structuré de manière à inclure les informations dans des balises <commentaire> et <bonus>.
- Structure XML attendue : Voici un exemple de la structure XML que vous devez générer :

```
<evaluation>
  <commentaire>Commentaires fusionnés ici...</commentaire>
  <bonus montant="total_bonus" raison="Fusion des évaluations" />
</evaluation>
```

Exécution de la requête : Après avoir configuré votre CTE et votre logique de transformation XML, exécutez la requête et affichez les résultats sous la forme d'un document XML fusionné.

Exercice 2 :

Une bibliothèque souhaite gérer les informations sur ses livres, mais elle doit prendre en compte des métadonnées complexes qui ne suivent pas une structure uniforme pour tous les livres. Par exemple, certains livres peuvent avoir plusieurs auteurs, d'autres peuvent être accompagnés de critiques provenant de plusieurs sources, et d'autres encore peuvent contenir des informations supplémentaires comme des traductions, des éditions spéciales, ou des chapitres supplémentaires. Ces informations peuvent varier d'un livre à l'autre, ce qui complique leur gestion dans une base de données relationnelle classique. Voici la structure hiérarchique des informations que nous souhaitons gérer dans la bibliothèque :

```
<livre>
  <id>1</id>
  <titre>Le Nom de la Rose</titre>
  <auteurs>
    <auteur>Umberto Eco</auteur>
  </auteurs>
  <chapitres>
    <chapitre>
      <titre>Le Premier Chapitre</titre>
      <page>1-25</page>
    </chapitre>
    <chapitre>
      <titre>Le Deuxième Chapitre</titre>
      <page>26-50</page>
    </chapitre>
  </chapitres>
  <critiques>
    <critique>
      <source>Le Monde</source>
      <note>3</note>
    </critique>
    <critique>
      <source>Le Figaro</source>
      <note>5</note>
    </critique>
  </critiques>
</livre>
```

Todo :

- En vous basant sur la structure du document XML ci-dessus, créez un fichier XSD (XML Schema Definition) qui définit le schéma des données pour un livre.
- Validez le document XML avec le XSD proposé dans l'outil suivant : https://www.xmlvalidation.com/
- Alimenter le document avec d'autres livres et écrire les requêtes XQuery pour :
    - Extraire le titre des livres dont la note moyenne des critiques est supérieure à 3.
    - Extraire les auteurs de livres dont le titre contient "Rose"
    - Extraire les livres avec au moins 2 chapitres
    - Calculer la somme des notes des critiques pour chaque livre
- Valider et vérifier les requêtes sur : https://www.videlibri.de/cgi-bin/xidelcgi

Annexe 1 :

Dans PostgreSQL, la fonction xpath() est utilisée pour extraire des données à partir de documents XML. Voici quelques explications des syntaxes les plus courantes que vous devrez utiliser pour interroger des données XML :

- Extraire un élément spécifique

Pour extraire un élément particulier d'un document XML, on utilise la syntaxe **//nom_element**. Par exemple, pour extraire l'élément <communication>, on utilisera la requête suivante :

```
xpath('//communication', details)
```

Cela renverra l'élément XML complet, c'est-à-dire `<communication>4</communication>`.

- Extraire le contenu texte d'un élément

Si vous souhaitez extraire uniquement la valeur du texte contenu dans un élément (et non l'élément XML lui-même), vous devez ajouter `/text()`. Par exemple :

```
xpath('//communication/text()', details)
```

Cette requête renverra la valeur "4" (le contenu texte de la balise <communication>), plutôt que l'élément XML complet.

- Extraire un attribut d'un élément

Si vous voulez récupérer un attribut d'un élément (par exemple, l'attribut montant dans `<bonus montant="1200" />`), vous devez utiliser la syntaxe `//@attribut`. Par exemple :

```
xpath('//bonus/@montant', details)
```

Cette requête renverra la valeur de l'attribut montant, ici "1200".

- Extraire un nombre

Pour obtenir un nombre à partir d'un attribut ou d'un élément XML, vous pouvez utiliser la fonction number() dans votre expression XPath. Par exemple :

```
xpath('number(//bonus/@montant)', details)
```

Cela convertira la valeur de l'attribut montant en un nombre et vous permettra de l'utiliser dans des comparaisons numériques.

- Obtenir un seul résultat d'une requête XPath

La fonction xpath() renvoie toujours un tableau de nœuds, même si vous vous attendez à un seul résultat. Pour récupérer le premier élément du tableau (par exemple, la première note), vous devez ajouter [1] et caster le résultat avec ::text (ou ::int, selon le type attendu). Par exemple :

```
SELECT (xpath('//communication/text()', details))[1]::text
```

Cela renverra la première valeur texte extraite de l'élément <communication>, que vous pourrez ensuite utiliser dans des requêtes SQL.

- Filtrer les résultats avec des conditions

XPath permet de filtrer les résultats en fonction de certaines conditions, mais cela peut être limité dans PostgreSQL. Par exemple, pour extraire des éléments avec une condition spécifique, vous pouvez utiliser une expression XPath comme :

```
xpath('//initiative[text()=5]', details)
```

Cela chercherait tous les éléments <initiative> dont le texte est égal à 5. Cependant, PostgreSQL ne permet pas directement ce type de filtrage dans xpath(), donc vous devrez parfois effectuer cette filtration dans la requête SQL, après l'extraction des données XML.

- Utiliser ILIKE avec XPath pour rechercher des mots-clés

Si vous voulez rechercher des mots-clés spécifiques dans le contenu d'un commentaire (par exemple, "projet X"), vous pouvez utiliser l'expression ILIKE dans PostgreSQL, qui permet une recherche insensible à la casse :

```
SELECT id_employe
FROM fiche_evaluation
WHERE (xpath('//commentaire/text()', details))[1]::text ILIKE '%projet X%'
```

Cela renverra les employés dont les commentaires contiennent le mot "projet X", en ignorant la casse.

Annexe 2

Quelques fonctions :

- **. string_agg()**

La fonction string_agg() permet d'agréger plusieurs valeurs de texte dans un seul champ, en les concaténant avec un séparateur.

```
string_agg(expression, separator)
```

expression : La colonne ou l'expression dont les valeurs doivent être concaténées.

separator : Le séparateur entre les valeurs concaténées (par exemple, un espace ' ').

Exemple d'utilisation :

```
string_agg(xpath('//commentaire/text()', details)::text, ' ') AS commentaires
```

Ici, nous utilisons xpath() pour extraire les commentaires sous forme de texte, puis nous les concaténons en une seule chaîne avec un espace comme séparateur.

- **.SUM()**

La fonction SUM() est utilisée pour additionner des valeurs numériques. Dans notre cas, on peut l'utilier pour additionner les montants des bonus extraits via xpath().

Exemple :

```
SUM((xpath('number(//bonus/@montant)', details))[1]::text::float) AS total_bonus
```

- **xmlconcat()**

La fonction xmlconcat() permet de concaténer plusieurs valeurs XML pour former un seul document XML.

Syntaxe :

```
xmlconcat(xml_expression1, xml_expression2, ...)
```

Exemple :

```
xmlconcat('<balise1>' || variable contenant les valuers || '</balise1>' ||
        '<balise2 attribut1="' || variable contenant les valeurs || '" attribut2="commentaire à écrire" />')
```

- **xmlparse()**

La fonction xmlparse() permet de convertir une chaîne de caractères en un document XML.

Syntaxe :

```
xmlparse(document 'xml_string')
```

Exemple :

```
xmlparse(document '<evaluation>' || ... || '</evaluation>')
```

Cela transforme la chaîne XML générée par xmlconcat() en un document XML valide.

Annexe 3 :

Qu'est-ce qu'une CTE ?

Une CTE (Common Table Expression), ou table virtuelle, est une requête temporaire définie dans le cadre de la requête principale. Elle vous permet de créer un sous-ensemble de résultats qui sera ensuite utilisé dans la requête principale. La CTE est définie avant la requête principale et ne persiste pas après l'exécution de cette dernière. Elle est particulièrement utile pour organiser des requêtes complexes en plusieurs étapes et améliorer leur lisibilité.

La CTE commence par le mot-clé WITH, suivi d'un nom pour la CTE (par exemple fiche_combinee), puis d'une requête qui renvoie les données nécessaires. Une fois définie, la CTE peut être utilisée comme une table normale dans la requête principale.

Syntaxe d'une CTE :

```
WITH nom_cte AS (
   -- Sous-requête ici
   SELECT ...
)
-- Requête principale qui utilise la CTE
SELECT ...
FROM nom_cte;
```