Université Libre de Bruxelles

INFO-H303

Bases De Données

Projet

Partie 2 : Dossier Patient Informatisé

Nieto Navarrete Matias , 502920
Vanderslagmolen Moïra , 547486
Ežerskis Andrius , 542698

B-INFO

Mai 2023

1 Méthode d'Extraction des Données

La méthode utilisée pour extraire les données comporte plusieurs étapes et est basée sur des scripts Bash et Python. Les données proviennent de différents fichiers au format SQL, XML ou CSV.

1.1 Création et importation de la base de données

Un script Bash est utilisé pour créer et configurer la base de données PostgreSQL. La base de données, nommée "dossier_medical", est d'abord supprimée si elle existe déjà, puis une nouvelle base de données est créée avec le même nom. Ensuite, un fichier SQL, "dossier_medical.sql", est importé dans la base de données créée qui permet de créer les tables.

1.2 Extraction et insertion des données XML

Le script Python "fileLoader.py" est utilisée pour extraire des données à partir de fichiers XML et les insérer dans la base de données. Chaque fonction gère un type d'élément spécifique du fichier XML, tel que "patient", "medecin", "pharmacien", "specialite" et "diagnostique". Chaque élément est analysé, et les informations sont extraites et insérées dans la base de données.

1.3 Extraction et insertion des données CSV

Le fichier "fileLoader.py" est aussi utilisée pour gérer l'extraction des données à partir des fichiers CSV. Ces fonctions prennent en charge l'insertion des données de "prescriptions", "pathologies" et "medicaments" dans la base de données à partir de leurs fichiers CSV respectifs. Pour chaque fichier, les données sont lues ligne par ligne, puis les informations sont extraites et insérées dans la base de données.

1.4 Gestion des conflits lors de l'insertion

Lors de l'insertion de données dans la base de données, il est possible que des conflits surviennent si une entrée avec la même clé primaire existe déjà. Pour gérer cela, nous utilisons l'instruction SQL "ON CONFLICT DO NOTHING" pour ignorer simplement l'insertion si un conflit se produit. Cette approche assure que la base de données reste cohérente et sans doublons.

2 Requêtes demandées

Les notations suivantes seront utilisées tout au long de cette section :

M : médicament
S : spécialité
P : prescription
PA : patient
PH : pathologie
SSA : spécialité_système_anatomique
MN : médecin
D : diagnostique
PS : pathologie_specialite

2.1 Requête 1

Cette requête sélectionne le nom commercial et le conditionnement des médicaments dont le DCI est spécifié. Elle trie les résultats par nom commercial et conditionnement, dans l'ordre croissant.

```sql
SELECT dci, nom_commercial, conditionnement -- On sélectionne les colonnes qui nous intéressent
FROM medicament -- On sélectionne la table qui nous intéresse
WHERE dci = 'YOUR_DCI' -- On filtre sur la DCI qui nous intéresse
ORDER BY nom_commercial ASC, conditionnement ASC; -- On trie par nom commercial puis par conditionnement
```

En algèbre relationnelle, cela s'exprime comme suit :

1. $temp1 \leftarrow \sigma_{dci='YOUR\_DCI'}(medicament)$
2. $Resultat \leftarrow \Pi_{dci,nom\_commercial,conditionnement}(temp1)$

En calcul relationnel des tuples, cela serait exprimé comme suit :

$$\{m.dci, m.nom\_commercial, m.conditionnement \mid m.dci =' YOUR\_DCI' \wedge M(m)\}$$

2.2 Requête 2

Cette requête sélectionne le nom des pathologies qui n'ont qu'une seule spécialité.

```sql
SELECT nom -- On sélectionne les colonnes qui nous intéressent
FROM pathologie -- On sélectionne la table qui nous intéresse
WHERE id_pathologie IN ( -- On filtre sur les pathologies qui nous intéressent
    SELECT id_pathologie -- On sélectionne la colonne qui nous intéresse
    FROM pathologie_specialite -- On sélectionne la table qui nous intéresse
    GROUP BY id_pathologie -- On groupe par id_pathologie
    HAVING COUNT(*) = 1 -- On filtre sur les pathologies qui n'ont qu'une seule spécialité
);
```

En algèbre relationnelle, cela s'exprime comme suit :

1. $temp1 \leftarrow \sigma_{count=1(id\_pathologie)}(PS)$
2. $temp2 \leftarrow (\Pi_{id\_pathologie}(temp1)$
3. $temp3 \leftarrow \sigma_{temp2.id\_pathologie \in PH.id\_pathologie}$
4. $temp4 \leftarrow \Pi_{nom}(temp3)$

En calcul relationnel des tuples, cela serait exprimé comme suit :

$$\{p.nom | P(p) \wedge \exists ps(PS(ps) \rightarrow p.id\_pathologie = (ps.id\_pathologie)\}$$

2.3 Requête 3

Cette requête sélectionne la spécialité médicale qui a le plus grand nombre de prescriptions.

```sql
SELECT s.nom -- On sélectionne les colonnes qui nous intéressent
FROM medecin m -- On sélectionne la table qui nous intéresse
INNER JOIN prescription p ON m.inami = p.inami_medecin -- On joint la table prescription
INNER JOIN specialite s ON m.id_specialite = s.id_specialite -- On joint la table specialite
GROUP BY s.nom -- On groupe par nom
ORDER BY COUNT(*) DESC -- On trie par nombre de prescriptions décroissant
LIMIT 1; -- On limite à 1 résultat
```

En algèbre relationnelle, cela s'exprime comme suit :

1. $temp1 \leftarrow (MN \bowtie_{MN.inami=P.inami\_medecin} (P))$
2. $temp2 \leftarrow (temp1 \bowtie_{temp1.id\_specialite=S.id\_specialite} (S))$
3. $\Pi_{nom}(temp2)$

En calcul relationnel des tuples, cela serait exprimé comme suit :

$$\{s.nom | MN(mn) \wedge S(s) \wedge P(p) \wedge mn.inami = p.inami\_medecin \wedge mn.id\_specialite = s.id\_specialite\}$$

2.4 Requête 6

Cette requête récupère les noms et numéros INAMI des médecins qui prescrivent des médicaments hors de leur spécialité, en listant les médicaments distincts qu'ils prescrivent.

```sql
SELECT medecin.inami AS medecin_inami, medecin.nom AS medecin_nom, -- On sélectionne les colonnes qui nous intéressent
  STRING_AGG(DISTINCT medicament.nom_commercial, ', ') AS medicament_noms
FROM prescription -- On sélectionne les tables qui nous intéressent
INNER JOIN medecin ON prescription.inami_medecin = medecin.inami -- On joint la table medecin
INNER JOIN medicament ON prescription.id_medicament = medicament.id_medicament -- On joint la table medicament
WHERE NOT EXISTS ( -- On fait une sous-requête pour exclure les médecins qui prescrivent des médicaments de leur spécialité
  SELECT NULL
  FROM specialite_systeme_anatomique ssa
  WHERE ssa.id_systeme_anatomique = medicament.id_systeme_anatomique
  AND ssa.id_specialite = medecin.id_specialite
)
GROUP BY medecin.inami, medecin.nom;
```

En algèbre relationnelle, cela s'exprime comme suit :

1. $temp1 \leftarrow (P \bowtie_{P.inami\_medecin=MN.inami} (MN))$
2. $temp2 \leftarrow (temp1 \bowtie_{temp1.id\_medicament=M.id\_medicament} (M))$
3. $temp3 \leftarrow (SSA \bowtie_{SSA.id\_specialite=MN.id\_specialite \wedge SSA.id\_systeme\_anatomique=M.id\_systeme\_anatomique} (temp2)$
4. $temp4 \leftarrow (\Pi_{NULL}(temp3))$
5. $temp5 \leftarrow (\Pi_{inami,nom}(temp2)$
6. $temp6 \leftarrow (\Pi_{MN.inami,MN.nom}(temp4 - temp5))$;
7. $temp7 \leftarrow (\alpha_{MN.inami:medecin\_inami}(temp6))$;
8. $\alpha_{MN.nom:medecin\_nom}(temp7)$;

En calcul relationnel des tuples, cela serait exprimé comme suit :

$$\{p.inami, p.nom | M(m) \wedge P(p) \wedge MN(mn) \wedge mn.inami = p.inami\_medecin \wedge p.id\_medicament = mn.id\_medicament\}$$

Nous n'avons pas représenté le *WHERE NOT EXISTS* en calcul tuple parce que il n'existe pas de notation correspondante.

2.5 Requête 8

Cette requête sélectionne le nom de la pathologie la plus couramment diagnostiquée.

```sql
SELECT nom, COUNT(nom) AS value_occurrence -- On compte le nombre de fois que le nom de la pathologie apparait
FROM diagnostique -- On selectionne la table diagnostique
INNER JOIN pathologie ON diagnostique.id_pathologie = pathologie.id_pathologie -- On fait une jointure avec la table pathologie
GROUP BY nom -- On groupe par nom
ORDER BY value_occurrence DESC -- On trie par ordre décroissant
LIMIT 1; -- On limite à 1
```

En algèbre relationnelle, cela s'exprime comme suit :

1. $temp1 \leftarrow (P \bowtie_{D.id\_pathologie=P.id\_pathologie} (D))$
2. $\Pi_{nom,COUNT(nom)}(temp1);$

En calcul relationnel des tuples, cela serait exprimé comme suit :

$$\{p.nom | D(d) \wedge P(p) \wedge d.id\_pathologie = p.id\_pathologie\}$$

2.6 Requête 9

Cette requête sélectionne les patients par leur NISS et leur nom, avec le nombre de médecins distincts qui leur ont fait une prescription.

```sql
SELECT patient.NISS, patient.nom, COUNT(DISTINCT prescription.inami_medecin) AS value_occurrence
FROM prescription -- On selectionne la table prescription
INNER JOIN patient ON prescription.NISS_patient = patient.NISS -- On fait une jointure avec la table patient
GROUP BY patient.NISS, patient.nom -- On groupe par NISS et nom
ORDER BY value_occurrence ASC; -- On trie par ordre croissant
```

En algèbre relationnelle, cela s'exprime comme suit :

1. $temp1 \leftarrow (P \bowtie_{NISS\_patient=NISS} PA))$
2. $\Pi_{NISS,nom}(temp1)$

En calcul relationnel des tuples, cela serait exprimé comme suit :

$$\{pa.NISS, pa.nom | PA(pa) \wedge P(p) \wedge pa.NISS = p.NISS\_patient\}$$

2.7 Requête 10

Cette requête sélectionne les médicaments qui n'ont pas été prescrits depuis une date spécifique.

```sql
SELECT medicament.nom_commercial, medicament.dci,MAX(prescription.date_prescription) AS derniere_prescription
FROM medicament -- On selectionne la table medicament
LEFT JOIN prescription ON medicament.id_medicament = prescription.id_medicament -- On fait une jointure avec la table prescription
GROUP BY  medicament.nom_commercial, medicament.dci -- On groupe par nom_commercial et dci
HAVING MAX(prescription.date_prescription) < DATE 'YOUR_DATE' OR MAX(prescription.date_prescription) IS NULL -- On filtre par date
ORDER BY medicament.nom_commercial; -- On trie par ordre croissant
```

En algèbre relationnelle, cela s'exprime comme suit :

1. $temp1 \leftarrow (M \rtimes_{M.id\_medicament=P.id\_medicament} P))$
2. $temp2 \leftarrow (\sigma_{derniere\_prescription<'YOUR\_DATE' \vee derniere\_prescription=NULL}(temp1))$
3. $Resultat \leftarrow \Pi_{nom\_commercial,dci,derniere\_prescription}(temp2)$

En calcul relationnel des tuples, cela serait exprimé comme suit :

$\{$m.nom_commercial, m.dci|M(m) $\wedge$ P(p) $\wedge$ m.id_medicament=p.id_medicament $\wedge \forall p_2(P(p_2) \rightarrow$ p_2.date_prescription $<$ (p.date_description) $\wedge$ p.date_description $<$ 'YOUR_DATE' $\}$

2.8 Justification

L'algèbre relationnelle et le calcul tuple pour la requête 7 n'a pas été réalisée car elle contient des éléments, tels que les clauses *CASE/END AS* et *MAX*. De plus, nous n'avons pas trouvé de moyens pour exprimer les clauses *order by* ou *group by*.

Pour notre application, lors de l'ajout d'un médecin, il est essentiel que sa spécialité existe déjà dans notre base de données. Sinon, le medecin ne sera pas ajouté.