Projet modélisation de bases de données

Arthur Gillier - Florian Chacun

Introduction

Ce projet a été réalisé dans le cadre du cours de Modélisation de Bases de données de l'Université de La Rochelle. Il a pour but de mettre en place une base de données relationnelle à partir de données issues de fichiers d'actes de mariage. Il faudra les nettoyer, les parser et les insérer de manière optimale dans une base de données PostgreSQL.

```
├── output
│   ├── actes.csv
│   ├── communes_id.csv
│   └── personnes_id.csv
├── scripts
│   └── insert_acte.py
├── source
│   ├── mariages.csv
│   └── mariages_L3.csv
└── sql
    ├── create.sql
    ├── drop.sql
    └── questions.sql
```

Conceptualisation et Modélisation des bases de données

La première étape de ce projet consiste à comprendre les données fournies et les modéliser de manière à répondre aux besoins initial.

Description des données

Les données sont extraites des archives départementales de Vendée et comprennent des informations détaillées sur les mariages enregistrés. Chaque enregistrement contenu dans le fichier mariages_L3_5k.csv comprend les colonnes suivantes :

1. Identifiant d’acte
2. Type d’acte
3. Nom personne A

4. Prénom personne A
5. Prénom père personne A
6. Nom mère personne A
7. Prénom mère personne A
8. Nom personne B
9. Prénom personne B
10. Prénom père personne B
11. Nom mère personne B
12. Prénom mère personne B
13. Commune
14. Département
15. Date
16. Num Vue

Définition des tables et des attributs

Sur la base de la description des données, nous avons identifié plusieurs entités : les actes, les personnes, les types d'actes, les communes et les départements. Chaque entité sera représentée par une table ou une énumération dans la base de données, avec des attributs correspondant aux différents champs des enregistrements.

Notre base de données sera donc représenté par ces tables :

- Acte : id, type, personne_a, personne_b, date, commune, num_vue
- Personne : id, nom, prenom, prenom_pere, nom_mere, prenom_mere
- Commune : id, departement, nom

Nous avons choisi de créer deux énumérations pour les types d'actes et les départements car nous avons un ensembles fini de valeurs possibles (les libelles pour les types et les numéros pour les départements). De plus cela offre des avantages en termes de cohérence des données, de facilité de maintenance et de lecture simplifiée.

- Departement : numéro
- Type : libelle

Il ne sera donc pas nécessaire d'avoir des contraintes de clés étrangères entre Acte/Type et Commune/Departement

Gestion des clés primaires et étrangères

Pour garantir l'intégrité et la cohérence des données, il faut définir les clés primaires et étrangères. Chaque table aura une clé primaire qui sera un identifiant unique. De plus, les relations entre les tables seront établies à l'aide de clés étrangères sur des liens logiques entre les différentes données.

Nous avons donc pour chaque table, un attribut id comme clé primaire.

Pour ce qui est des clés étrangères :

- Acte :

  “personne_a” et “personne_b” associés à la table Personne

  “commune” associé à la table Commune

Relations entre les tables

Dans notre schéma de base de données pour les registres de mariages, nous avons identifié et choisit d'utiliser une realtion Many-to-One.

Relation Many-to-One entre la Table "acte" et la Table "personne"

Chaque acte peut impliquer deux personnes distinctes : la personne A et la personne B. Pour modéliser cette relation, "personne_a" et "personne_b" servent de clés étrangères faisant référence aux identifiants uniques des personnes dans la table "personne".

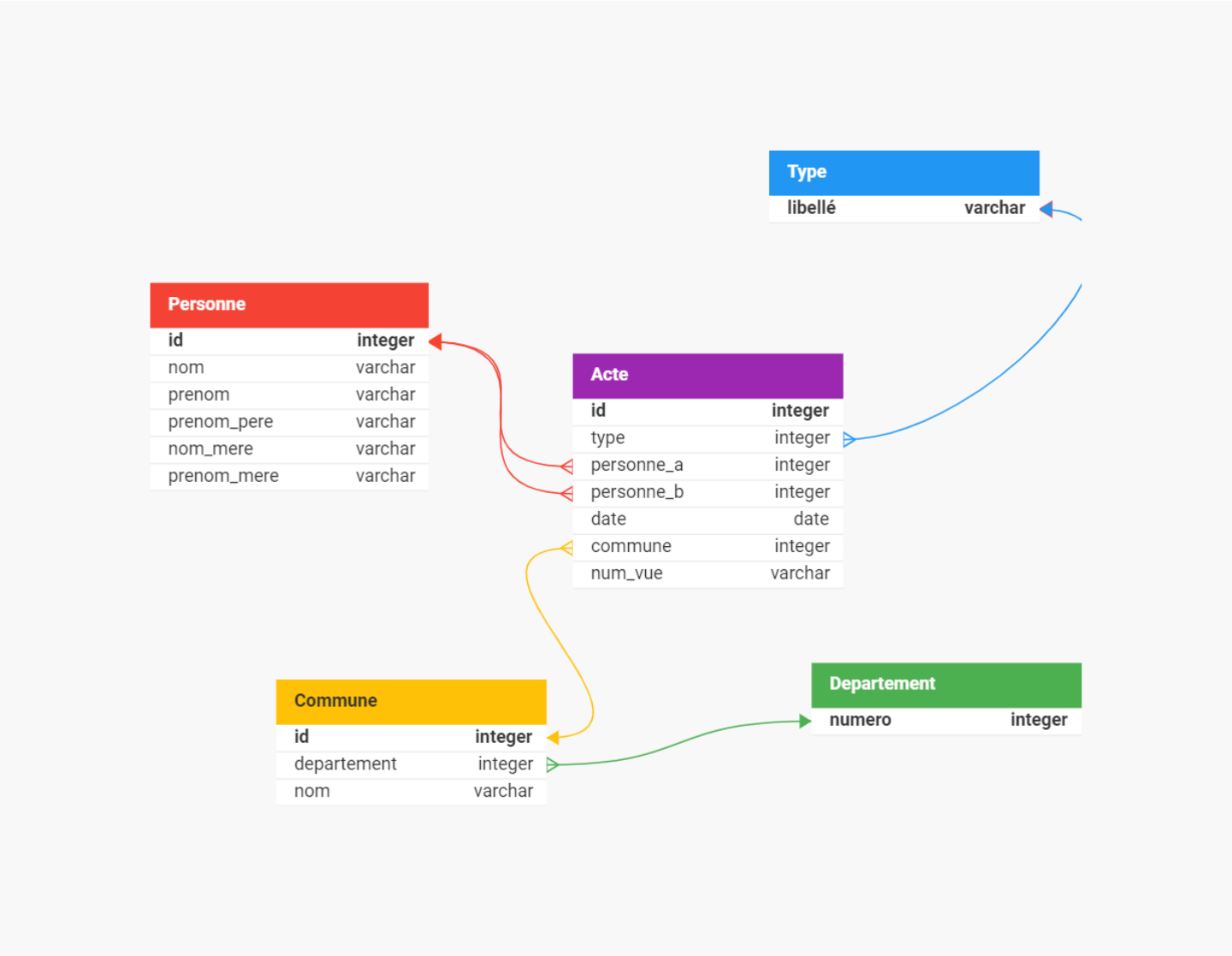
Cette relation many-to-one entre les tables "acte" et "personne" permet de représenter le lien entre les actes enregistrés et les personnes dans chaque acte. Un acte de mariage peut avoir deux personnes impliquées (personne_a et personne_b), et chaque personne peut être liée à plusieurs actes.

Relation Many-to-One entre la Table "acte" et la Table "commune"

Chaque acte de mariage est enregistré dans une commune. Pour représenter cette association, la colonne "commune" dans la table "acte" est une clé étrangère faisant référence aux identifiants uniques des communes dans la table "commune".

Cette relation many-to-one entre les tables "acte" et "commune" permet de lier chaque acte de mariage à la commune où il a été enregistré. Un acte peut être enregistré dans une seule commune, mais plusieurs actes peuvent être enregistrés dans la même commune. Cela facilite la recherche et l'analyse des actes de mariage selon leur lieu d'enregistrement.

Schéma de la base de données



Notez que les tables `departement` et `type` ne sont pas réellement des tables mais des enums.

Normalisation du schéma relationnel

Principes de la normalisation

La normalisation est le processus qui vise à réduire la redondance des données, à améliorer l'intégrité et à minimiser les anomalies lors de la manipulation des données. Nous nous sommes basés sur les formes normales, dont les principales sont la première forme normale (1NF), la deuxième forme normale (2NF), et la troisième forme normale (3NF).

Pour appliquer les principes de la normalisation à notre base de données, nous avons examiné la structure de chaque table et les dépendances fonctionnelles entre les attributs. Nous avons fait en sorte que chaque table soit en 1NF en nous assurant que chaque attribut est atomique.

Une relation est en 1FN si tous les attributs :

- Sont atomiques (pas décomposables) / non répétitifs
- (Sont constants dans le temps)

Ensuite, nous avons vérifié que chaque table était en 2NF en s'assurant que chaque attribut non-clé dépendait entièrement de la clé primaire.

Une relation 1FN est en 2e forme normale si :

- Aucun attribut non-clé ne dépend que d'une partie d'une clé
- Elle ne concerne que les tables à clé primaire composée

Par exemple, dans notre table "Acte", les attributs "type", "personne_a" et "personne_b" dépendent directement de l'identifiant de l'acte. Enfin, nous avons examiné la 3NF pour nous assurer qu'aucun attribut non-clé ne dépendait transitivement d'un autre attribut non-clé.

Une relation 2FN est en 3e forme normale si :

- Il n'existe aucune DF entre les attributs non-clés

Par exemple, dans notre table "Commune", l'attribut "departement" dépend directement de l'identifiant de la commune, sans dépendre du "nom" de la commune. Afin d'éviter la redondance, nous avons créé des tables pour des valeurs qui se répétaient comme par exemple la commune et les personnes qui seront/peuvent être constants ou se répéter. De plus, la gestion des départements et des types d'actes en énumération évite encore plus la redondance.

Création des tables

Une fois PostgreSQL configuré, nous avons pu créer nos tables de base de données en spécifiant les noms des colonnes, les types de données, les contraintes de clé primaire et étrangère en utilisant le modèle relationnel que nous avons conçu.

Nous avons veillé à ce que la structure de nos tables corresponde exactement à notre modèle relationnel, en tenant compte de toutes les relations et contraintes définies précédemment. Les clés primaires sont directement définies lors de la création.

Département

```
CREATE TYPE departement AS ENUM ('44', '49', '79', '85');
```

Type

```
CREATE TYPE type_acte AS ENUM ('Certificat de mariage', 'Contrat de mariage', 'Divorce', 'Mari
```

Commune

```
CREATE TABLE IF NOT EXISTS commune(
    id INT PRIMARY KEY,
    nom VARCHAR(255) NOT NULL,
    departement departement NOT NULL
)
```

Personne

```
CREATE TABLE IF NOT EXISTS personne (
    id INT PRIMARY KEY,
    nom VARCHAR(255),
    prenom VARCHAR(255),
    prenom_pere VARCHAR(255),
    nom_mere VARCHAR(255),
    prenom_mere VARCHAR(255)
);
```

Acte

```
CREATE TABLE acte(
    id INT PRIMARY KEY,
    type_id type_acte NOT NULL,
    personne_a INT,
    personne_b INT,
    commune INT,
    date_ TIMESTAMP WITH TIME ZONE,
    num_vue VARCHAR(255) DEFAULT 'null'
)
```

Découpage des données

En suivant la méthode donnée nanani nanana

Découpage du csv :

```
# Découpage du fichier mariage_L3_5k.csv pour obtenir les communes et les personnes.
cut -f13,14 -d ',' mariages_L3_5k.csv | sort | uniq > commune.csv
cut -f3,4,5,6,7 -d ',' mariages_L3_5k.csv | sort | uniq > personnes.csv

# Notons que le >> permet de rajouter les données à la fin du fichier sans écraser le contenu
cut -f8,9,10,11,12 -d ',' mariages_L3_5k.csv | sort | uniq >> personnes.csv

# Ajout des id pour chaque ligne des csv de personne et commune.
awk -F, '{print NR","$0}' personne.csv > personne_id.csv
awk -F, '{print NR","$0}' commune.csv > commune_id.csv
```

Récupération des actes

Afin de récupérer les actes nous avons utiliser un script python qui utilise les fichiers `mariages_L3_5k.csv` , `personne_id.csv` et `commune_id.csv` pour générer un fichier `actes.csv` qui contient les actes de mariage.

```
import pandas as pd

# Charger le fichier personnes.csv dans un DataFrame
personnes_df = pd.read_csv('/media/Qi/agillier/L3/Projet-Modélisation/data/personne_id.csv', h
commune_df = pd.read_csv('/media/Qi/agillier/L3/Projet-Modélisation/data/commune_id.csv', head

# Fonction pour rechercher le numéro dans personnes.csv
def trouver_id_personne(nom, prenom, prenom_pere, nom_mere, prenom_mere):
    filtre = (personnes_df['nom'] == nom) & (personnes_df['prenom'] == prenom) & (personnes_df
    resultats = personnes_df[filtre]
    if not resultats.empty:
        return resultats['id'].values[0]
    else:
        return None

def trouver_id_commune(nom, departement):
    filtre = (commune_df['nom'] == nom) & (commune_df['departement'].astype(str) == str(depart
    resultats = commune_df[filtre]
    if not resultats.empty:
        return resultats['id'].values[0]
    else:
        return None


# Ouvrir un fichier de sortie CSV pour écrire les résultats
with open('/media/Qi/agillier/L3/Projet-Modélisation/data/mariages_L3_5k.csv', 'r') as f:
    with open('/media/Qi/agillier/L3/Projet-Modélisation/data/actes.csv', 'w') as output_file:
```

```
        output_file.write("Identifiant d’acte,Type d’acte,Id Personne A,Id Personne B,Commune,
        for line in f:
            mariage_info = line.strip().split(',')
            if len(mariage_info) == 16:  # Assurez-vous que la ligne contient suffisamment de
                nom_personne_a = mariage_info[2]
                prenom_personne_a = mariage_info[3]
                prenom_pere_personne_a = mariage_info[4]
                nom_mere_personne_a = mariage_info[5]
                prenom_mere_personne_a = mariage_info[6]
                id_personne_a = trouver_id_personne(nom_personne_a, prenom_personne_a, prenom_
                nom_personne_b = mariage_info[7]
                prenom_personne_b = mariage_info[8]
                prenom_pere_personne_b = mariage_info[9]
                nom_mere_personne_b = mariage_info[10]
                prenom_mere_personne_b = mariage_info[11]
                id_personne_b = trouver_id_personne(nom_personne_b, prenom_personne_b, prenom_
                commune = trouver_id_commune(mariage_info[12], mariage_info[13])
                temps = mariage_info[14].split('/')
                if len(temps) == 3:
                    mariage_info[14] = temps[2] + '-' + temps[1] + '-' + temps[0]
                else:
                    mariage_info[14] = ''
                if (id_personne_a is not None and id_personne_b is not None):
                    output_file.write(f"{mariage_info[0]},{mariage_info[1]},{id_personne_a},{i
```

Insertion des données dans la base de données :

```
COPY personne FROM 'C:\Program Files\PostgreSQL\16\mariages\personne_id.csv' DELIMITER ',' CSV

/* Notons que l'on utilise le CSV HEADER afin d'éliminer le header du csv. */
COPY acte (id,type_id,personne_a,personne_b,commune,date_,num_vue) FROM 'C:\Program Files\Post

COPY commune (id,nom,departement) FROM 'C:\Program Files\PostgreSQL\16\mariages\commune_id.csv
```

Ajout des relations

Acte -> Commune

```
ALTER TABLE acte ADD CONSTRAINT acte_fk3 FOREIGN KEY (commune) REFERENCES commune(id);
```

Acte -> Personne (a) & (b)

```
/* Acte -> Personne (a) */
ALTER TABLE acte ADD CONSTRAINT acte_fk1 FOREIGN KEY (personne_a) REFERENCES personne(id);
/* Acte -> Personne (b) */
ALTER TABLE acte ADD CONSTRAINT acte_fk2 FOREIGN KEY (personne_b) REFERENCES personne(id);
```

Requêtes

La quantité de communes par département

```
SELECT departement, COUNT(*) AS nombre_de_communes
FROM commune
GROUP BY departement;
```

Résultat :

	departement departement	nombre_de_communes bigint
1	79	51
2	49	2
3	85	313
4	44	9

La quantité d'actes à LUÇON

```
SELECT COUNT(*) AS nombre_d_actes
FROM acte
INNER JOIN commune ON acte.commune = commune.id
WHERE commune.nom = 'LUÇON';
```

Résultat : `105`

La quantité de “contrats de mariage” avant 1855

```
SELECT COUNT(*) AS nombre_de_contrats_de_mariage
FROM acte
WHERE type_id = 'Contrat de mariage' AND date_ < '1855-01-01';
```

Résultat : `196`

La commune avec la plus quantité de “publications de mariage”

```
SELECT commune.nom AS commune,
       COUNT(*) AS nombre_de_publications_de_mariage
FROM acte
INNER JOIN commune ON acte.commune = commune.id
WHERE acte.type_id = 'Publication de mariage'
GROUP BY commune.nom
ORDER BY COUNT(*) DESC
LIMIT 1;
```

Résultat : `SAINT PIERRE DU CHEMIN : 20`

La date du premier acte et le dernier acte

```
SELECT MIN(date_) AS premiere_date_acte,
       MAX(date_) AS derniere_date_acte
FROM acte;
```

Résultat : `Première date : "1581-12-23 00:00:00+00:09:21"` & `Dernière date : "1915-09-14 00:00:00+00"`

Conclusion

Tout est fonctionnel pour le fichier de 5k lignes, pour passer sur le fichier à 500k lignes il faudrait ajouter une table département et une table type d'actes. Il faudrait également modifier le script python pour qu'il prenne en compte ces nouvelles tables.

Auteurs

- Arthur Gillier
- Florian Chacun
- Sujet : Université de La Rochelle.