



Cette figure représente le schéma Entité-Relation de la base de données, elle a été générée avec la base de données. Ce schéma montre les différentes entités de la base de données, ainsi que les relations entre elles. Chaque entité est représentée par un rectangle, et chaque relation est représentée par une ligne qui relie les entités concernées. Les clés primaires et étrangères sont également indiquées pour chaque entité, permettant ainsi de comprendre la structure de la base de données et les liens entre les différentes tables.

## 2.1 Description du script Peuplement

Peuplement sert à parcourir et traiter le fichier XML contenant les informations sur les employés d'Enron. Voici une explication détaillée de ce que fait chaque partie de votre script :

### 1. Importation des modules et configuration Django :

- Vous importez les modules nécessaires pour travailler avec XML (`xml.etree.ElementTree`) et pour configurer Django (`os` et `django`).
- Vous configurez l'environnement Django en spécifiant le fichier de configuration des paramètres.

### 2. Parcours du fichier XML :

- Vous utilisez `ET.parse()` pour charger le fichier XML et `getroot()` pour obtenir la racine de l'arbre XML.
- Vous parcourez ensuite chaque élément (employé) dans la racine.

### 3. Traitement des employés :

- Pour chaque employé, vous créez une instance de la classe `Employee` du modèle Django.
- Vous initialisez également une liste `mails` pour stocker les adresses e-mail associées à cet employé.
- Vous parcourez ensuite chaque élément (attribut) de l'employé et effectuez les actions suivantes :
  - Si l'attribut est une adresse e-mail, vous créez une instance de la classe `MailAddress`, stockez l'adresse e-mail et l'ajoutez à la liste `mails`.
  - Si l'attribut est le nom de famille, le prénom ou la boîte aux lettres, vous mettez à jour les attributs correspondants de l'instance de `Employee`.
- Une fois que vous avez parcouru tous les attributs de l'employé, vous enregistrez l'instance de `Employee` dans la base de données Django.
- Vous parcourez ensuite la liste `mails`, enregistrez chaque adresse e-mail dans la base de données Django et créez des instances `EmployeeToMailaddress` pour associer chaque adresse e-mail à l'employé correspondant.

## 2.2 Description du script Mail Passer

C'est un analyseur de courriels pour extraire des informations des fichiers de courriels dans une structure de base de données. Voici une explication détaillée de ce que chaque fonction fait dans votre script :

### 1. `mailParser(file_path)` :

- Cette fonction principale est utilisée pour extraire des informations à partir d'un fichier de courriel.
- Elle ouvre le fichier, lit son contenu, puis appelle d'autres fonctions pour extraire des informations spécifiques telles que la date, le sujet et l'expéditeur du courriel.

- Elle vérifie également si le courriel a déjà été vu en appelant la fonction `deja_vu()` pour éviter les doublons.
- Elle enregistre ensuite les informations extraites dans une instance de la classe `Message` et les lie aux destinataires du courriel en appelant la fonction `getReceivers()`.
- Elle appelle également la fonction `defineMessageType()` pour définir le type de message (interne, externe, mixte) en fonction des destinataires.
- Enfin, elle appelle la fonction `populateInteractions()` pour enregistrer les interactions entre l'expéditeur et les destinataires dans la table `Interactions`.

## 2. Fonctions auxiliaires :

- `get_entete(raw)` : Extrait l'en-tête du courriel.
- `internalMailCheck(adresse)` : Vérifie si une adresse e-mail est interne.
- `deja_vu(testdate, testsubject, testsender)` : Vérifie si un courriel est déjà présent dans la base de données.
- `getJmId(raw)` : Extrait l'identifiant JavaMail du courriel.
- `getDate(raw)` : Extrait la date du courriel.
- `getSubject(raw)` : Extrait le sujet du courriel.
- `getSender(raw)` : Extrait l'expéditeur du courriel.
- `handleAddress(address_to_handle)` : Gère l'adresse e-mail pour créer une instance de `MailAddress`.
- `getReceivers(raw, local_address)` : Extrait les destinataires du courriel.
- `defineMesssageType(receivers)` : Définit le type de message en fonction des destinataires.
- `populateInteractions(sender, receivers, date, message)` : Enregistre les interactions entre l'expéditeur et les destinataires.

## 3. Boucle principale :

- La boucle principale parcourt tous les fichiers dans le répertoire `maildir` et appelle `mailParser()` pour chaque fichier.

## 4. Nettoyage :

- À la fin du script, il affiche le temps d'exécution.

# 3 Application

La partie visuelle de notre application Django utilise des codes HTML se trouvant dans un dossier *Templates*, les fichiers `urls.py`, `views.py` et `forms.py`. Voici une explication des principaux composants du code :

## 3.1 urls.py

Le fichier `urls.py` définit les chemins d'accès (URL) de l'application et associe chaque chemin à une vue spécifique. Voici une explication détaillée des chemins d'accès définis dans ce fichier :

- **accueil** : L'URL `"/accueil"` est associée à la fonction de vue `accueil`. Cette vue affiche la page d'accueil de l'application, fournissant des statistiques sur le nombre de courriels et de messages.
- **employees** : L'URL `"/employees"` est associée à la fonction de vue `employees_table`. Cette vue affiche une table répertoriant tous les employés d'Enron.

- **basicmining** : L'URL `"/basicmining"` est associée à la fonction de vue `basic_mining`. Cette vue permet aux utilisateurs de rechercher des courriels en fonction de divers critères tels que la plage de dates, le type de courriel, l'expéditeur, etc.
- **message/<int :message\_id>/** : L'URL `"/message/<int :message_id>/"` est associée à la fonction de vue `show_message`. Cette vue affiche le contenu d'un courriel spécifique identifié par son identifiant unique (`message_id`).
- **seuils** : L'URL `"/seuils"` est associée à la fonction de vue `seuils`. Cette vue affiche une table répertoriant les employés ayant envoyé ou reçu un nombre de courriels supérieur à un seuil donné.
- **interactions** : L'URL `"/interactions"` est associée à la fonction de vue `interactions`. Cette vue affiche une table répertoriant les interactions entre les employés, avec des options pour filtrer les données.
- **conversation** : L'URL `"/conversation/<int :employee_a_id>-<int :employee_b_id>/<str :fromDate>/<str :toDate>"` est associée à la fonction de vue `conversation`. Cette vue affiche une conversation entre deux employés sur une période donnée.
- **achalandage** : L'URL `"/achalandage"` est associée à la fonction de vue `achalandage`. Cette vue affiche un graphique d'affluence des courriels sur une période donnée, avec des options pour filtrer les données.
- **mailmatcher** : L'URL `"/mailmatcher"` est associée à la fonction de vue `mailmatcher`. Cette vue affiche un formulaire permettant de faire correspondre des employés à des adresses e-mail.

## 3.2 views.py

Le fichier `views.py` contient les fonctions de vue qui traitent les requêtes HTTP et renvoient les réponses correspondantes. Voici un résumé des fonctionnalités fournies par les vues dans `views.py` :

- **accueil** : Affiche la page d'accueil de l'application, avec des statistiques sur le nombre de courriels et de messages.
- **employees\_table** : Affiche une table répertoriant tous les employés d'Enron.
- **basic\_mining** : Permet aux utilisateurs de rechercher des courriels en fonction de divers critères comme la plage de dates, le type de courriel, l'expéditeur, etc.
- **show\_message** : Affiche le contenu d'un courriel spécifique.
- **seuils** : Affiche une table répertoriant les employés ayant envoyé ou reçu un nombre de courriels supérieur à un seuil donné.
- **interactions** : Affiche une table répertoriant les interactions entre les employés, avec des options pour filtrer les données.
- **conversation** : Affiche une conversation entre deux employés sur une période donnée.
- **achalandage** : Affiche un graphique d'affluence des courriels sur une période donnée, avec des options pour filtrer les données.
- **mailmatcher** : Affiche un formulaire permettant de faire correspondre des employés à des adresses e-mail.

## 3.3 forms.py

Ce fichier définit les formulaires utilisés dans l'application. Ces formulaires permettent à l'utilisateur d'entrer des données et de les soumettre via des requêtes POST. Voici une explication des formulaires définis dans ce fichier :

- **Basic\_mining\_form** : Ce formulaire est utilisé dans la vue `basic_mining` pour permettre aux utilisateurs de spécifier des critères de recherche pour les courriels, tels que la plage de dates, le type de courriel, l'expéditeur, etc.
- **Seuils\_form** : Ce formulaire est utilisé dans la vue `seuils` pour permettre aux utilisateurs de spécifier des critères de seuil pour afficher les employés ayant envoyé ou reçu un nombre de courriels supérieur à un seuil donné.
- **Interactions\_form** : Ce formulaire est utilisé dans la vue `interactions` pour permettre aux utilisateurs de spécifier des critères pour afficher les interactions entre les employés, tels que la plage de dates, le seuil d'interaction, etc.
- **Achalandage\_form** : Ce formulaire est utilisé dans la vue `achalandage` pour permettre aux utilisateurs de spécifier des critères pour afficher le graphique d'affluence des courriels sur une période donnée, tels que la plage de dates et le type de courriel.
- **Mailmatcher\_form** : Ce formulaire est utilisé dans la vue `mailmatcher` pour permettre aux utilisateurs de faire correspondre des employés à des adresses e-mail.