Projet Affaire Enron

Auderic et Aziz

Mai 2024

Table des matières

1 Contexte

Le scandale Enron, ou affaire Enron, est un cas de fraude et de manipulation financière découvert en 2001, qui s'est soldé par la faillite de l'entreprise Enron, un temps septième capitalisation des États-Unis, et par le démantèlement et la disparition de facto de son auditeur Andersen. Il s'agit à l'époque de la plus grande faillite de l'histoire américaine (à côté de la faillite scandaleuse d'une autre grande entreprise américaine WorldCom à peu près à la même époque et pour des raisons similaires). - Source : Wikipedia

Il s'est avéré que l'analyse des échanges de mails effectués entre les employés pouvait être pertinente pour mener une enquête sur cette affaire.

Le but de ce projet est l'implémentation d'une application web permettant à un enquêteur d'analyser les échanges effectués entre les employés d'Enron, et ceci en passant des requêtes paramétrées à travers une interface web.

2 Base de données

On part d'un fichier XML où on peut trouver les informations associés à un nombre d'employés de l'entreprise (nom, prenom, catégorie, boîte mail, adresses mail). De plus, on dispose d'un dossier "maildir" contenant des boîtes mail sous forme de dossiers. Chaque mailbox est propre à un employé et contient elle même des dossiers qui eux même contiennent des fichiers .txt représentant les mails envoyés ou reçus par l'employé, formatés selon le protocole SMTP. Ce formatage est intéressant puisqu'il nous a permis par la suite d'extraire les métadonnées propres à chaque mail (expediteur, destinataire(s), objet, date et heure) et son contenu.

2.1 Modèle de données

Nous avons commencé par mettre en place un modèle de données adapté aux informations dont on dispose. Au vu des données disponibles dans le fichier XML, nous avons pensé qu'il était judicieux de créer une table Personne modélisant les employés. Dans cette table, nous avons un identifiant unique comme clé primaire, les attributs nom, prénom, catégorie, boîte mail. Comme un employé peut avoir plusieurs adresses mails, nous avons décidé de mettre en place une table Email, caractérisée par un identifiant unique comme clé primaire, l'adresse mail et la clé étrangère de l'employé associé. Ceci implique une relation de cardinalité 1 - N entre les tables Personne et Email. Ensuite, nous avons exploré brièvement les fichiers du dossier maildir, et au vu de leur structure, nous avons mis en place une table Message caractérisée par un identifiant unique, les attributs expediteur_id (qui est l'identifiant de l'email

de l'expéditeur passé en clé étrangère), objet, date et heure et le contenu. On obtient donc une relation 1 - 1 entre les tables Message et Email. De même, un message peut avoir plusieurs destinataires, ceci nous a amené à créer une table Destinataire, contenant un identifiant unique pour chaque destinataire, l'attribut emails_id (l'identifiant de l'email du destinataire passé en clé étrangère) et l'attribut messages_id qui est l'identifiant du message reçu comme clé étrangère, ce qui crée une relation 1 - 1 entre les tables Destinataire et Email et une relation 1 - N entre les tables Destinataire et Message.

2.2 Peuplement

L'implémentation de notre projet sous la forme de projet Django nous a permis de créer des algorithmes Python pour peupler la base de donnée, utilisant les objets créés dans le fichier models.py.

Dans un premier temps, pour peupler la table Personne avec le fichier XML des employés, nous avons mis en place un script "peuplement_xml.py" qui utilise la bibliothèque "xml.etree.ElementTree" permettant de parser le fichier, tout en se basant sur les balises associées à chaque attribut. Dans un second temps, le peuplement des tables Message et Destinataire a nécessité un script réparti sur trois fichiers "extrait_infos.py" qui est le script contenant les fonctions permettant d'extraire les informations associées à chaque mail, le fichier "fonctions_peuplement_maildir.py" qui contient les fonctions permettant de peupler un message dans la base de données, et le fichier "peuplement_maildir.py" qui parcoure tout le dossier "maildir" et alimente la base de données avec la totalité des données.

2.3 Difficultés rencontrées

En parcourant les fichiers des mails, nous avons constaté qu'il y avait plusieurs adresses mail de personnes dont les attributs étaient inconnus. Nous avons donc créé une personne "bidon" nodata qui possède les valeur "N/A" dans tous ses attributs de la table Personne. Cependant, parmi ces adresses mail, certaines étaient d'Enron ("@enron") et d'autres étaient externes. Pour les différencier, nous avons ajouté l'attribut "est_interne" dans la table Email, qui prend les valeurs "Y" si l'adresse mail est de l'entreprise et "N" sinon.

3 Application

Dans la partie Application du projet, le but était de concevoir une interface pour l'enquêteur, sous forme de formulaire lui permettant de paramétrer ses requêtes. Pour chaque requête, nous avons mis en place une fonction Python qui prend en argument les paramètres de l'utilisateurs et qui utilisant psycopg , réalise une requête SQL sur la base de données selon les paramètres de la fonction (sauf la requête 6 qui a été codée intégralement en Python), et retourne le résultat de cette requête. Comme chaque requête doit être passée à travers un formulaire, nous avons profité de la fonctionnalité de Django qui permet de créer les formulaires sous la forme d'objets forms.Form et de les afficher facilement grâce au format JSON utilisé dans le template de l'affichage du formulaire. Dans le fichier "views.py", chaque requête a été associée à une vue qui affiche la formulaire de la vue (contenu dans le fichier templates sous la forme "form_requete.tmpl"), capture les réponses de l'utilisateur et les fait passer en paramètre dans la fonction de la requête en question. Ensuite, le résultat est affiché grâce au template "resultat_requete.tmpl" qui utilise le format JSON pour itérer sur les résultats retournés par la fonction et les afficher convenablement. Une vue pour la page d'accueil a également été définie, et cette page affiche la totalité des requêtes disponibles avec des liens hypertexte dirigeant vers les vues des requêtes.

Liste des requêtes

Parmi les 7 requêtes proposées dans le sujet, nous avons réussi à faire les 6 premières requêtes. En effet, nous n'avons pas réussi à modéliser les conversations à cause d'une difficulté à identifier les mails appartenant à une même conversation.

```
Requete_1
# si l'utilisateur choisit nom

SELECT * FROM monappli_personne AS personne
INNER JOIN
    (SELECT *
    FROM monappli_email
    WHERE monappli_email.personne_id IN
        (SELECT monappli_personne.id
            FROM monappli_personne
            WHERE monappli_personne.nom = '{nom}')) AS email
ON personne.id = email.personne_id;
```

```
# si l'utilisateur choisit adresse mail

SELECT * FROM
    (SELECT *
    FROM monappli_personne
    WHERE id =
        (SELECT personne_id
            FROM monappli_email
            WHERE monappli_email.adresse_mail = '{adresse_mail}')
    ) AS personne
INNER JOIN
    monappli_email AS email
ON
    personne.id = email.personne_id;


Requete_2
# Si l'utilsateur choisit Reçus,signe,x,Date_inf,Date_sup ; signe = {>|<}
SELECT
    p.nom,
    p.prenom,
    e.adresse_mail,
    SUM(CASE WHEN m.type = 'interne' THEN 1 ELSE 0 END) AS nb_mail_internes_envoyes,
    SUM(CASE WHEN m.type = 'interne-externe' THEN 1 ELSE 0 END) AS nb_mail_externes_envoye
    COUNT(*) AS nb_mail_envoyes_total
FROM monappli_personne p
JOIN monappli_email e ON p.id = e.personne_id
JOIN monappli_message m
            ON e.id = m.expediteur_id WHERE m.date_heure >= '{Date_inf}'
            AND m.date_heure < '{Date_sup}' AND e.est_interne='Y'
GROUP BY p.nom, p.prenom, e.adresse_mail
HAVING COUNT(*) {signe} {x}
ORDER BY COUNT(*) DESC;

# Si l'utilsateur choisit Envoyés,signe,x,Date_inf,Date_sup  ; signe = {>|<}
SELECT
    p.nom,
    p.prenom,
    e.adresse_mail,
    SUM(CASE WHEN m.type = 'interne' THEN 1 ELSE 0 END) AS nb_mail_internes_recus,
    SUM(CASE WHEN m.type = 'interne-externe' THEN 1 ELSE 0 END) AS nb_mail_externes_recus,
```

```
    COUNT(*) AS nb_mail_recus_total
FROM monappli_personne p
JOIN monappli_email e ON p.id = e.personne_id
JOIN monappli_destinataire d ON d.emails_id = e.id
JOIN monappli_message m ON m.id = d.messages_id
WHERE (m.date_heure >= '{Date_inf}' AND m.date_heure < '{Date_sup}' AND e.est_interne='Y')
GROUP BY p.nom, p.prenom, e.adresse_mail
HAVING COUNT(*) {signe} {x}
ORDER BY COUNT(*) DESC;


# Si l'utilsateur choisit Reçus+Envoyés,signe,x,Date_inf,Date_sup  ; signe = {>|<}

SELECT
    nom,
    prenom,
    adresse_mail,
    SUM(nb_mail_internes_envoyes + nb_mail_internes_recus) AS nb_mail_internes_total,
    SUM(nb_mail_externes_envoyes + nb_mail_externes_recus) AS nb_mail_externes_total,
    SUM(nb_mail_internes_envoyes + nb_mail_internes_recus + nb_mail_externes_envoyes + nb_
FROM (
    SELECT
        p.nom,
        p.prenom,
        e.adresse_mail,
        SUM(CASE WHEN m.type = 'interne' THEN 1 ELSE 0 END) AS nb_mail_internes_envoyes,
        SUM(CASE WHEN m.type = 'interne-externe' THEN 1 ELSE 0 END) AS nb_mail_externes_en
        0 AS nb_mail_internes_recus,
        0 AS nb_mail_externes_recus
   FROM monappli_personne p
    JOIN monappli_email e ON p.id = e.personne_id
    JOIN monappli_message m ON e.id = m.expediteur_id
    WHERE m.date_heure >= '{Date-inf}' AND m.date_heure < '{Date_sup}' AND e.est_interne='
    GROUP BY p.nom, p.prenom, e.adresse_mail

    UNION ALL

    SELECT
        p.nom,
        p.prenom,
        e.adresse_mail,
```

```
        0 AS nb_mail_internes_envoyes,
        0 AS nb_mail_externes_envoyes,
        SUM(CASE WHEN m.type = 'interne' THEN 1 ELSE 0 END) AS nb_mail_internes_recus,
        SUM(CASE WHEN m.type = 'interne-externe' THEN 1 ELSE 0 END) AS nb_mail_externes_re
    FROM monappli_personne p
    JOIN monappli_email e ON p.id = e.personne_id
    JOIN monappli_destinataire d ON d.emails_id = e.id
    JOIN monappli_message m ON m.id = d.messages_id
    WHERE (m.date_heure >= '{Date_inf}' AND m.date_heure < '{Date_sup}' AND e.est_interne=
    GROUP BY p.nom, p.prenom, e.adresse_mail
) AS mails
GROUP BY nom, prenom, adresse_mail
HAVING nb_mail_total {signe} {x}
ORDER BY nb_mail_total DESC;
```

```
Requete_3
# Lutilisateur choisit le nom,prenom de l'employé et l'intervalle de temps.

SELECT * FROM (
    SELECT *
    FROM monappli_personne
    WHERE monappli_personne.id IN (
        SELECT monappli_email.personne_id
        FROM monappli_email
        WHERE monappli_email.id IN (
            SELECT monappli_destinataire.emails_id
            FROM monappli_destinataire
            WHERE monappli_destinataire.messages_id IN (
                SELECT id
                FROM monappli_message
                WHERE expediteur_id IN (
                    SELECT id
                    FROM monappli_email
                    WHERE monappli_email.personne_id = (
                        SELECT id
                        FROM monappli_personne
                        WHERE nom = '{nom}' AND prenom = '{prenom}'
                    ))
                    AND monappli_message.date_heure >= '{date_heure_min}'
```

```
                    AND monappli_message.date_heure <= '{date_heure_max}'
            )
            AND monappli_email.est_interne = 'Y'
        )
    )

   UNION

SELECT *
FROM monappli_personne p
WHERE p.id IN (
    SELECT personne_id
    FROM monappli_email
    WHERE id IN (
        SELECT expediteur_id
        FROM monappli_message
        WHERE (
            monappli_message.id IN (
                SELECT messages_id
                FROM monappli_destinataire
                WHERE emails_id IN (
                    SELECT id
                    FROM monappli_email
                    WHERE personne_id = (
                        SELECT id
                        FROM monappli_personne
                        WHERE nom = '{nom}'
                        AND prenom = '{prenom}'
                    )
                )
            )
            AND
            (monappli_message.date_heure > '{date_heure_min}'
            AND monappli_message.date_heure < '{date_heure_max}')
        )
    )
    AND est_interne = 'Y'
)) as personne
INNER JOIN
monappli_email as email ON (personne.id = email.personne_id AND email.est_interne='Y');
```

```
Requete_4
# L'utilisateur choisit la peridode [Date_inf, Date_sup] et le seuil.
SELECT p1.nom as nom_expediteur, p1.prenom as prenom_expediteur,
       p2.nom as nom_destinataire, p2.prenom as prenom_destinataire,
       COUNT(*) as nombre_mails_echanges
FROM monappli_personne p1
JOIN monappli_email em1 ON p1.id = em1.personne_id
JOIN monappli_message m ON em1.id = m.expediteur_id
JOIN monappli_destinataire d ON m.id = d.messages_id
JOIN monappli_email em2 ON d.emails_id = em2.id
JOIN monappli_personne p2 ON em2.personne_id = p2.id
WHERE m.date_heure >= '{Date_inf}' AND m.date_heure <= '{Date_sup}'
AND p1.id != 150 AND p2.id != 150
GROUP BY p1.id, p2.id
HAVING COUNT(*) >= {seuil}
ORDER BY COUNT(*) DESC;

Requete_5

# Lutilisateur choisit la periode [Date_inf,Date_sup]

SELECT
date(date_heure),
COUNT(*) AS total_mails,
SUM(CASE WHEN type = 'interne' THEN 1 ELSE 0 END) AS mails_internes,
SUM(CASE WHEN type = 'interne-externe' THEN 1 ELSE 0 END) AS mails_interne_externe
FROM monappli_message WHERE date(date_heure) >= '{Date_inf}' AND date(date_heure) <= '{Dat
GROUP BY date
ORDER BY COUNT(*) DESC;

Requete_6
# L'utilisateur choisit la liste de mots et la présentation et le mode (par expediteur ou

def requete6(mots, mode):
    resultat = []
    pattern = "|".join(mots)

    messages_ids = [message.id for message in Message.objects.all() if re.search(pattern,
```

```
if mode == "expediteur":
    expediteurs_messages = defaultdict(list)

    for message in Message.objects.filter(id__in=messages_ids):
        expediteurs_messages[message.expediteur_id].append(message.id)

    for expediteur_id, message_ids in expediteurs_messages.items():
        expediteur_email = Email.objects.filter(id=expediteur_id).first()
        if expediteur_email:
            for message_id in message_ids:
                message = Message.objects.get(id=message_id)
                if all(word in message.contenu for word in mots):
                    resultat.append((message_id, expediteur_email.adresse_mail))

elif mode == "destinataire":
    destinataires_messages = defaultdict(list)

    for destinataire in Destinataire.objects.filter(messages_id__in=messages_ids):
        destinataires_messages[destinataire.emails_id].append(destinataire.messages_id

    for email_id, message_ids in destinataires_messages.items():
        email = Email.objects.filter(id=email_id).first()
        if email:
            for message_id in message_ids:
                message = Message.objects.get(id=message_id)
                if all(word in message.contenu for word in mots):
                    resultat.append((message_id, email.adresse_mail))

return resultat
```