Alexandre HERGAUX BASES DE DONNÉS

Tom BERNE--GALLIERI

Paul BERNARDIN

# Rapport de Projet : Big Brother

Rendu le 30 novembre 2024

Année académique 2024 / 2025

Une image contenant capture d’écran, logo, symbole, conception

Description générée automatiquement

Fait dans le cadre d’un projet du cours de Base de Données

École d’ingénieur Denis Diderot

**Introduction :**

Je suis sûr que vous vous êtes déjà interrogé sur ce qu’il est possible de faire avec toutes les données collectées chaque seconde à travers le monde. Certains parleront de théorie du complot, mais à travers ce mini-projet, nous allons vous démontrer qu’en organisant judicieusement un ensemble de bases de données, il est possible de créer un logiciel d’espionnage autonome. Ce logiciel interagirait avec des fichiers de données pour enrichir et structurer son arborescence.

Notre projet s’inspire directement du personnage fictif **Big Brother**, imaginé par George Orwell dans son roman *1984*.

En pratique, l’idée est simple : nous souhaitons récupérer et traiter des données collectées au sein d’une population. L’objectif est de croiser ces données avec celles que nous possédons déjà afin de constituer une base de données riche, cohérente et exploitable. Cette démarche vise à faciliter l’accès à des informations sensibles dans le but d’exercer une forme de pression sur la population, permettant ainsi de servir certains intérêts.

Pour la mise en application, nous prendrons comme point de départ une table intitulée **“personne”**, qui constituera le centre autour duquel graviteront toutes les données importées. Chaque information manipulée devra être soit un entier, soit un réel. Les informations textuelles seront remplacées par des clés étrangères, qui pointeront vers leurs valeurs réelles stockées dans des tables de définition. Ces tables de définition permettent de centraliser certaines informations, comme des listes de noms ou des adresses de rues, tout en optimisant l’utilisation de la mémoire.

Les données seront importées sous forme de fichiers CSV. Une fois comparées aux données existantes, de nouvelles tables seront générées pour contenir des informations plus spécifiques, et les données manquantes seront ajoutées aux enregistrements déjà présents dans la base.

Pour aller plus loin, nous avons jugé pertinent d’ajouter une fonctionnalité permettant de consulter toutes les modifications effectuées sur la base de données. Ainsi, une seconde couche de notre programme consistera en une gestion globale et automatisée des logs. Cette fonctionnalité permettra de retracer l’historique des changements et d’assurer un suivi complet des manipulations réalisées.

**Problématique :**

**Comment centraliser, normaliser et intégrer automatiquement des données provenant de sources diverses tout en garantissant leur cohérence et leur unicité dans une base de données relationnelle ?**

**Cahier des charges :**

PROJET Big Brother

Le but est de créer et manipuler une base de données dynamique à l’aide du langage de programmation PSQL. Cette base à pour but de réunir des informations fictives sur des personnes et de s’enrichir au fil du temps à l’aide de fichiers qu’on lui fournira.

Notre objectif à court terme est de réussir à coder un algorithme qui nous permette de lire des données sur un fichier CSV, de les interpréter et de les comparer avec les données déjà présente dans la base. Il se chargera de compléter les informations des fiches personnes et de créer une nouvelle table si on obtient une information qui n’a pas de lien avec des tables déjà existante. L’algorithme devra également être capable de proposer des logs pour permettre un suivit détailler des modifications de table.

La base de données s’agencera autour de différents grands axes : La donnée interne, la donnée externe, l’intégration et la gestion des logs.

**1)Les données internes ou le cœur de la base de données :**

Créer et remplir des tables de définitions qui servirons de pieds d’appuis pour tous les liens qui seront ensuite réalisé par l’algorithme. Ces tables initiales seront au nombre de 6,

* Définir et remplir une table de référence « PRENOM\_\_REF »

Elle fournit un identifiant unique pour chacun des prénoms.

* Définir et remplir une table de référence « NOM\_\_REF »

Elle fournit un identifiant unique pour chacun des noms.

* Définir et remplir une table de référence « VILLE\_\_REF »

Elle fournit un identifiant unique pour chacune des villes françaises.

* Définir et remplir une table de référence « RUE\_\_REF »

Elle fournit un identifiant unique pour chacune des rues, et l’identifiant de la ville dans laquelle elle se situe.

* Définir et remplir une table de référence « ADRESSE\_\_REF »

Elle fournit un identifiant unique pour chacune des adresses ainsi que l’identifiant de la ville et de la rue dans laquelle elle se situe.

* Définir et remplir une table de référence « GENRE\_\_REF »

Id 1 pour un homme et 2 pour une femme.

Ces 6 tables serviront de référence pour pouvoir manipuler des Integer au lieu d’informations textuelles.

Le cœur de notre projet réside sur le fait de fiché les citoyens français, la BD devra donc nous fournir une table PERSONNE composé d’attributs figés qui permettent de définir l’identité d’un citoyen de manière unique.

* Définir la table « PERSONNE » qui doit contenir un prénom, un nom, une date de naissance, une ville de naissance et un genre.

Cette table personne et ses attributs ne suffisent pas pour avoir toutes les informations nécessaires sur un citoyen. La base de données devra donc proposer 4 tables supplémentaires qui apporteront des informations complémentaires mais néanmoins essentielles sur les citoyens. Ces informations peuvent quant à elles évoluer dans le temps, c’est pourquoi nous les distinguons des données figées directement liées à la table PERSONNE.

Une des premières choses qui pourrait nous intéresser sur quelqu’un serait son adresse, la BD dois fournir une table liée à personne avec les adresses postales des citoyens.

* Définir la table « ADRESSE\_POSTALE » qui doit lier une adresse postale à un ou plusieurs individus, cette table sera composée d’un id\_adresse qui fera référence à une adresse, d’un id\_personne représentant une personne, et d’une date de début et de fin d’investiture des lieux.

La base de données devra également contenir une table qui lie une personne à un numéro de téléphone.

* Définir la table « NUMERO\_TEL », cette table sera composée d’un id\_personne représentant une personne, et d’un numéro de téléphone sous forme d’un integer.

La base de données devra contenir une table qui lie une personne à une adresse mail.

* Définir la table « ADRESSE\_MAIL », cette table sera composée d’un id\_personne représentant une personne, et d’une adresse mail sous forme d’un id référencé dans une table de définition

Enfin, la base de données devra contenir une table qui lie une personne à un réseau sociale.

* Définir la table « RESEAUX », cette table sera composée d’un id représentant une personne et d’un id lié à un réseau social dans une table de définition.

**2) La donnée externe :**

La base de données doit être capable de recevoir et de traiter de l’information sous forme de fichiers CSV.

Plus concrètement, la base doit être capable de lire le fichier CSV et de créer une table temporaire qui contient toutes les informations du CSV. Ensuite, la base de données devra fournir une fonction capable de comparer les informations de cette table temporaires avec les données déjà présente dans la base de manière efficace pour finalement enrichir cette dernière et compléter les fichages.

La structure du CSV soumis à la base doit respecter la typographie suivante :

* La première ligne du fichier nous renseigne sur les attributs et de leur type de la table à créer. (header)

Le nom des attributs vient en premier puis un espace nous indique la fin du nom, vient ensuite le type de l’attribut et enfin une virgule le sépare de l’attribut suivant.

« nom TEXT,prenom TEXT,… »

* Chacune des lignes suivantes représente un tuple où les données sont également séparées par une virgule et où les valeurs coïncident aux attributs du header.

« Martin,Alexandre,… »

**3) L’intégration :**

La base de données devra fournir une fonction appelée « import\_fichier » qui permet d'importer un fichier CSV dans une table cible. Cette fonction prendra en entrée le chemin relatif du fichier CSV et le nom de la table cible. Elle devra être capable d’effectuer les étapes suivantes :

* Lire l'en-tête du fichier CSV pour récupérer les attributs avec et sans types.
* Créer une table temporaire pour importer les données du fichier CSV.
* Normaliser les données importées
* Vérifier et créer les tables de référence nécessaires pour les attributs spécifiques.
* Remplir ces tables de référence avec les données uniques.
* Créer la table cible avec les colonnes appropriées.
* Remplir la table cible avec les données transformées de la table temporaire.
* Supprimer la table temporaire après traitement.

Ensuite, vient le traitement de cette table importée, la base de données devra donc fournir une fonction appelée.

« lier\_personne\_import », permettant de lier automatiquement les enregistrements d'une table importée à la table

« Personne ». Elle prend en entrée le nom de la table importée et compare les colonnes d'identité (nom, prénom, date de naissance, etc.). Si une correspondance est trouvée en fonction d'un seuil de similarité arbitrairement choisi, l'enregistrement importé est associé à la personne existante. Sinon, une nouvelle entrée est créée dans la table

« Personne ». Cette approche garantit la cohérence des données et évite les doublons.

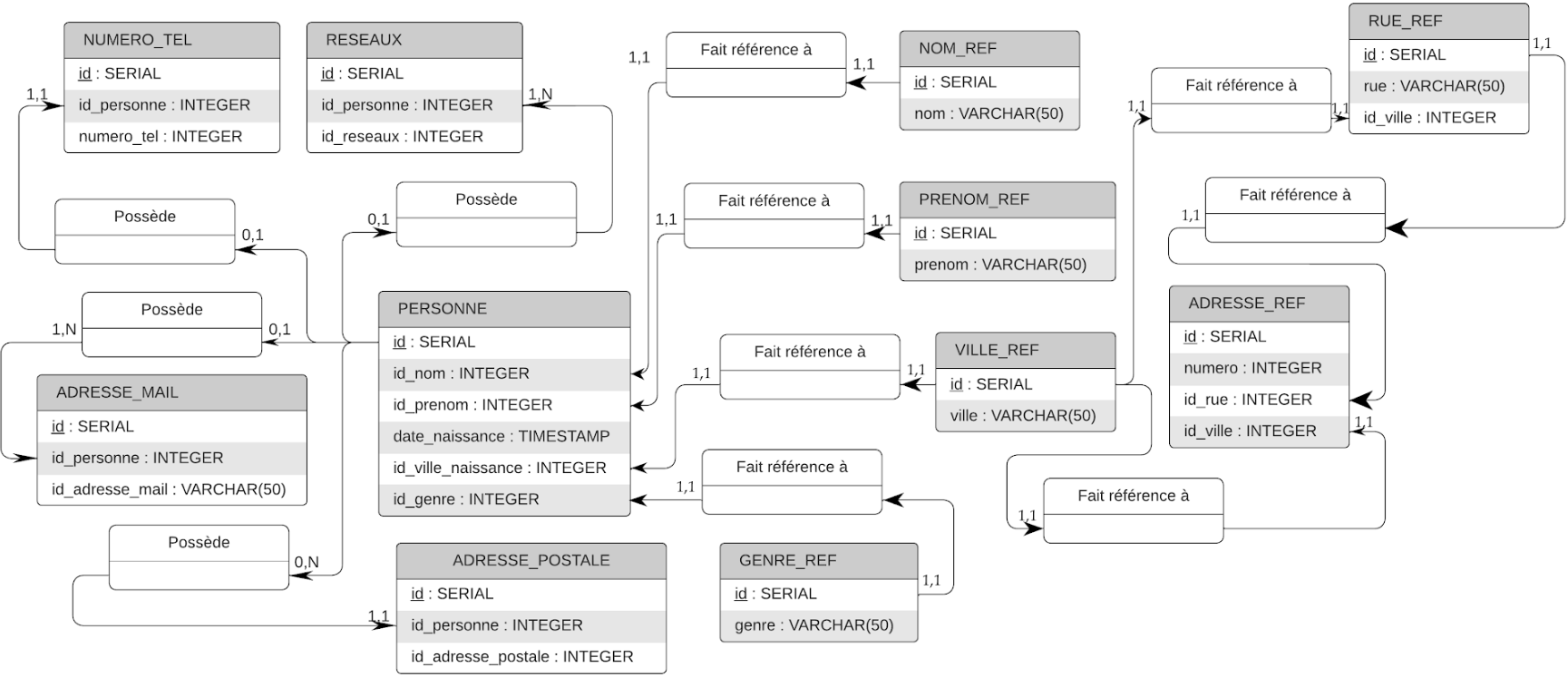
**4) La gestion des logs :**

Pour permettre un suivi des évolutions de situation de nos citoyens, mais aussi avoir un contrôle de sécurité sur les modifications effectuées sur les données interne de notre base de données, cette dernière devra associer à chacune des tables de données interne une table miroir dites « audit ».

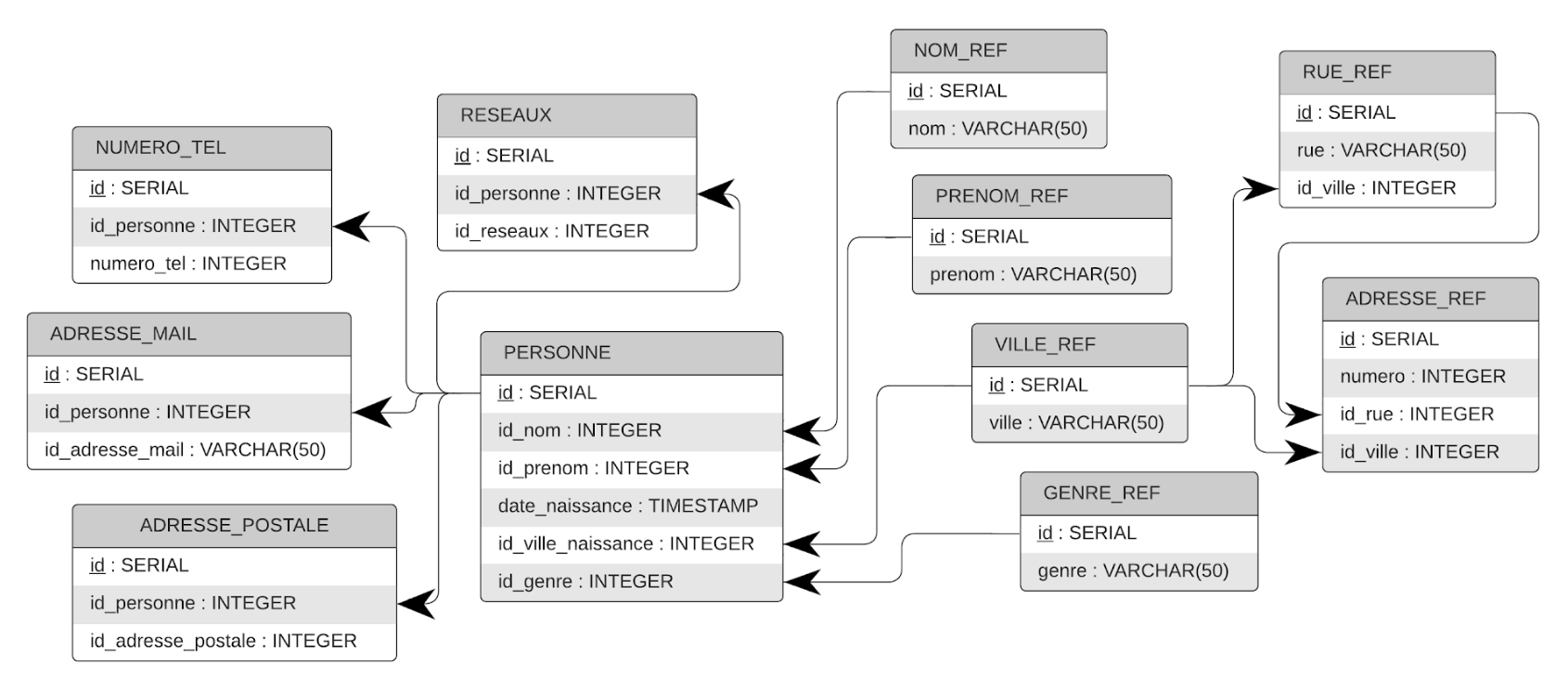
Ainsi l’algorithme devra implémenter, pour chacune des tables « PRENOM\_\_REF, NOM\_\_REF, VILLE\_\_REF, RUE\_\_REF, ADRESSE\_\_REF, GENRE\_\_REF, PERSONNE, ADRESSE\_POSTALE, NUMERO\_TEL, ADRESSE\_MAIL, RESEAUX » une table miroir « nom\_de\_la\_table\_audit » dans laquelle sera enregistrer pour chacun des tuples les modifications effectuées, le nom de l’administrateur qui les a modifiés, l’heure de la modification et qui garde en mémoire la valeur précédente avant modification.

Ainsi, pour chacune des tables internes, la base de données devra être munie d’une fonctions « audit\_ changes » qui s’appuiera sur l’utilisation de triggers pour détecter les modifications et réaliser toutes les actions précédemment demandées. Ces fonctions devront être appelées systématiquement à chaque modification des tables correspondantes.

**MCD :**



**Modélisation des tables :**



**Réponse technique :**

Nous allons ici vous détailler les solutions techniques que nous avons trouvé pour répondre à la problématique. Penchons-nous sur le premier point qui n’est rien d’autre que la centralisation des données.

**Centralisation des données :**

Pour répondre à ce problème, deux implémentations majeures sont nécessaires, l’importation des données et la création automatique de nouvelles entrées.

Attardons-nous sur l’importation des données :

La fonction import\_fichier réalise l’importation des données depuis un fichier CSV jusqu’à la table cible créer ou déjà existante. Cependant il y a de nombreuses étapes, le fonctionnement de import\_fichier et diviser en plusieurs sous fonctions. Tout d’abord elle lit le fichier source puis créer une table temporaire avec la fonction « create\_table\_temp() » puis elle insert les données brut dans cette table. Ensuite, pour chaque attribut de notre table temporaire, si son type est du texte, alors nous créons une nouvelle table de référence nommée « nom\_attribut\_\_ref » composée d’un id et d’un attribut nom\_attribut ; pour tout autre type on ne fait rien. Enfin, nous créons la table finale accueillant les données, tous les attributs étant du texte auparavant sont remplacés pour les id de leurs tables de référence, les autres types restes telle quel (comme DATE, INTEGER, REAL).

Jusqu’ici, la table n’as pas de réel lien avec les données déjà intégrer, or le but de ce projet est d’arriver à créer du lien avec toutes les données afin d’obtenir une connaissance absolu.

Tout d’abord nous créons une nouvelle colonne id\_personne attendant un id de la table personne. Si avec les données que l’on détient de la nouvelle table suffisent à identifier une personne déjà implémentée, on importe l’id dans la colonne id\_personne de notre nouvelle table. Sinon, la sous fonction « créer\_nouvelle\_personne\_from\_import » est appelé et elle construit dynamiquement la requête d’insertion.

C’est ici qu’intervient la création automatique de nouvelles entrées, une des notions nouvelle et importante dont nous ne pouvons-nous passer ici est la manipulation de chaîne de caractère que nous utilisons partout dans notre code. C’est une notion essentielle pour la création d’une base dynamique qui va devoir générer des requêtes à partir d’éléments abstraits. Nous expliquerons le processus de comparaison dans la partie 3 de la réponse technique car elle a un lien plus fort avec la notion de cohérence des données.

**Normalisation des données :**

Un autre point important de notre problématique concerne la normalisation des données, en effet un tel algorithme ne serait pas robuste sans une normalisation systématique des données enregistrées. Cela poserait des problèmes pour comparer les données mais également pour toutes les fonctions qui manipule des données abstraites mais destinés à un type bien défini.

Sur ce sujet une des fonctions les plus importante est sans doute la fonction « normalize\_table\_contain » qui parcourt toutes les tables de référence et les tables importantes, qui s’arrête à chaque fois quelle détecte un attribut TEXT et normalise le texte en retirant les accents, remplaçant tous les espaces et les tirets par des underscores de la touche 8.

Ainsi on se rend compte que le choix d’avoir opter pour des tables de référence s’avère être payant quand il s’agit de normalisation des données car c’est le seul endroit où il faut agir sur la normalisation de la forme, tout le reste de la base de données exploite des clés étrangères sous forme d’integers faisant référence aux vrais données encodées dans ces tables de références. De plus, les données normalisées sont réparties dans des tables de référence spécifiques via des mécanismes d'insertion conditionnelle ce qui assure une vérification de la normalisation, dans le cas contraire elle ne serait même pas acceptée dans les tables de référence.

**Cohérence et unicité des données :**

Dans cette partie nous allons voir ce qui se cache derrière « l’intelligence » de la base de données, ou autrement comment l’algorithme fait pour réaliser des choix probent. Pour la gestion des conflits d’unicité nous utilisons récurremment la clause « ON CONFLICT DO NOTHING », cela permet d’empêcher les duplications et de maintenir la cohérence des données. Imaginez une base où pour chaque information redécouverte sur une personne on ajoute un clone de cette même personne, cela rendrai le modèle ingérable. L’autre secret un peu plus astucieux réside dans la fonction « lier\_personne\_import » chargé de décider « si une personne est bien une personne » au moment de lier la nouvelle table et les données préexistante. L’algorithme va décider si les informations fraichement importées correspondent à une personne déjà enregistré dans la table de référence. On peut déjà souligner l'utilisation des tableaux qui permet une gestion dynamique et générique des colonnes et des attributs, facilitant ainsi l'intégration automatique des données importées. Nous pouvons également souligner l’importance de l’apprentissage de l’utilisation des tables « informations\_schema » utiliser partout pour assurer une abstraction totale de notre BD. Cependant la magie prend effet durant le petit algorithme improvisé que nous avons implémenté. Elle suis la loi suivante :

Admettons qu’une possible correspondance a été trouvé en comparant les différents attributs de la table personne et de la nouvelle table.

Soit n, le nombre d’attribut en correspondance avec les attributs de la table « personne », on considère que la correspondance est acceptable si :

Cette règle peut être intéressante car elle est robuste face aux différences d’échantillons. Néanmoins, cette solution reste imparfaite car il y’a toujours une faible probabilité qu’une personne ait les mêmes informations. Dans le cadre de ce mini-projet nous pouvons admettre cette incertitude.

Comme mentionné au début de ce rapport, les attributs de la table **“personne”** ont été choisis parce qu’ils changent relativement peu au cours de la vie d’un individu, constituant ainsi une base solide pour lier les autres données à celle-ci. Cependant, d’autres informations, telles que l’adresse postale, l’adresse e-mail et les réseaux sociaux, sont tout aussi essentielles pour connecter une personne au monde actuel, fortement numérique.

C’est ici qu’intervient la fonction « insert\_into\_corresponding\_table », qui joue un rôle clé dans la cohérence de la base de données. Cette fonction est une méthode importante de généralisation des données. Par exemple, lorsqu’un tuple provenant d’un fichier CSV est associé à une donnée importante, la fonction vérifie si les informations contenues dans ce tuple peuvent enrichir les tables pertinentes liées à cette personne.

Cette fonction est cruciale pour la généralisation de certaines données cruciales à l’identification et à la liaison d’une personne. Nous aurions également pu l’évoquer dans la partie dédiée à la centralisation des données, tant son rôle est déterminant pour garantir une structure organisée et évolutive.

**Audit et traçabilité des modifications :**

La dernière spécificité de notre base de données est qu’elle possède un historique de chacune des opérations effectuer sur toutes les tables. Il y a deux grands avantages à enregistrer toutes ces modifications, premièrement cela assure une traçabilité des utilisateurs qui modifierait la base de données, secondement ces informations s’avère précieuse pour retracer la vie des personnes et constater des évolutions dans leur vie personnelle. Pour réaliser cette fonctionnalité nous appelons la fonction « create\_\_audit\_\_changes » du fichier fonctions\_manip, celle-ci va créer une table d’audit puis générer la fonction du triger\_audit qui donnera les instructions au trigger et enfin elle créera le triger\_audit. Il sera chargé de détecter chacune des modifications effectuées sur la table à laquelle il est lié et lèvera les informations de l’utilisateur qui l’a modifié telle que la nouvelle information, l’ancienne et la date à laquelle la modification a été effectué.

On comprend qu’une telle implémentation ouvre un panel important d’action dans l’analyse de l’évolution des situations professionnelles ou personnelles des citoyens. Nous n’avons pas pour l’instant exploré ces pistes.

**Conclusion :**

La solution proposée centralise les données importées, les normalise en identifiant les champs pertinents, et les intègre dans la base de données en garantissant unicité et cohérence. Elle renforce également la gestion de la qualité des données grâce à des mécanismes d'audit et de détection des doublons.

**Commentaires, problèmes rencontrés et améliorations :**

Nous avons rencontré plusieurs problèmes lors du développement de notre projet. Tout d’abord, il a été difficile de déterminer quelles étaient les tables minimales nécessaires et celles maximales possibles à l’initialisation de la base, afin de garantir une extension efficace. Ensuite, l’une des principales complexités résidait dans le choix des approches pour normaliser, architecturer et établir les liens entre les données. L’agencement des données pouvant prendre de nombreuses formes, nous avons également dû nous accorder sur le type de données que nous allions recevoir. Dans notre cas, il s’agit exclusivement de tuples liés à des personnes, sans inclure de fichiers relatifs aux entreprises ou aux institutions publiques. Enfin, un défi majeur a été de savoir comment reconnaître si deux personnes aux informations similaires (par exemple, nom, prénom, date de naissance) étaient en réalité la même personne. Pour gérer ce problème sans prendre de décisions trop rigides, nous avons choisi de nous appuyer sur une règle statistique pour déterminer les correspondances.

Accepter des fichiers contenant des informations autres que celles liées aux personnes, comme des données sur les entreprises ou les partis politiques, permettrait d’étendre considérablement la connectivité de notre base de données. Cela reviendrait à relier notre base actuelle à d’autres bases similaires mais axées sur des domaines différents. Pour y parvenir, il aurait fallu développer un nouvel algorithme capable de lier des noms de tables, comme “entreprise”, avec leurs attributs respectifs, tout en identifiant les similarités entre les informations. Cet algorithme remplacerait ensuite les valeurs par des clés étrangères faisant référence aux entreprises correspondantes. Cependant, notre base repose sur un algorithme complexe et dynamique, ce qui rend difficile la prise en compte de tous les cas possibles. Certains scénarios spécifiques pourraient poser des problèmes de robustesse, nécessitant une optimisation future. Par ailleurs, après réflexion, nous nous interrogeons sur l’utilisation des tables de référence. Bien qu’elles simplifient la gestion des doublons et des comparaisons, elles rendent la manipulation et les requêtes sur la base plus complexe, ce qui pourrait nuire à la flexibilité globale du système.

**Manuel de fonctionnement de la base de données :**

* Changer le chemin relatif dans la fonction import\_fichier
* Se placer dans le répertoire bigbrother
* Lancer le script init\_setup.sql
* Réaliser ensuite les requêtes de votre choix