Anonymiser des données avec PostgreSQL

Qui suis-je ?

Thomas Reiss

Membre de l'association PostgreSQLfr
Utilisateur de PostgreSQL depuis 2004

Contexte

Synthétiquement, à travers la Loi Informatique et Libertés, il nous est imposé de ne pas travailler sur avec des données à caractère nominatif dans différents cas, notamment :

- sur les jeux de données de développements, ;

- sur les jeux de données de formation ;

- etc.

Contexte

Pour palier cela, on peut générer un jeu de données à partir de zéro, ou procéder à l' « *anonymisation* » de données déjà existantes.

L'étape d'anonymisation doit faire perdre le caractère nominatif des données, sans remettre en question le sens des données en terme métier (un nom reste un nom).

Contexte

Besoin interne d'anonymiser des bases de données hétérogènes issues d'une même application, avec possibilité de refaire une opération identique sur d'autres données.

L'offre existante

Deux produits éditeurs disponibles :
- Data Masker de Cortina
- Data Masking Pack d'Oracle

Ces produits ne fonctionnent qu'avec une base de données Oracle, *« ça coûte »* !

Aucune alternative libre trouvée, mais il y en a peut-être...

Les différentes méthodes

Par **suppression** : on supprime ou tronque la donnée, elle perd tout son sens en terme métier ;

Par **mélange** : on mélange les données originales, elles perdent leur caractère nominatif mais pas leur sens.
Certainement le meilleur moyen pour converser la qualité des données ;

Par **remplacement** : on remplace la donnée réelle par une donnée fictive choisie aléatoirement dans un référentiel, les données ont perdu leur caractère nominatif ainsi que leur « qualité ».

La solution utilisée

La solution proposée s'appuie sur deux composants :

Un ETL pour charger les données métier dans la base de données de travail ;

Une base de données PostgreSQL, contenant un référentiel et des procédures stockées pour aider l'anonymisation.

Le référentiel

Une table « referentiel » :

```
CREATE TABLE referentiel
(
  id serial NOT NULL, -- PK, clé technique
  valeur text, -- Valeur de reference
  "type" text, -- Type de donnees de reference
  seq integer, -- clé « métier » ou surrogate key
  fk text, -- FK: référence à la valeur
  fk_type text - FK: rérérence au type
);
```

Le référentiel

Constitué de différents types de données :

- Noms patronymiques ;
- Prénoms ;
- Départements ;
- Référentiel des communes INSEE ;
- Référentiel d'adresses relatif au précédent ;

La table de remplacement

```
CREATE TABLE remplace_ano (
"type" text,
ancienne text,
nouvelle text
);
```

type = type de données référentielle ;
ancienne = valeur d'origine à remplacer ;
nouvelle = valeur de remplacement correspondante ;

Avantage: conserve une répartition des données identique.

Les autre types de données

Les adresses sont gérées dans le même esprit, mais c'est un peu plus compliqué.

D'autres types de données nominatives existent, mais peuvent être générées/modifiées à partir des données issues du référentiel :

- Les dates sont légèrement modifiées ;
- Les numéros de sécurité sociale sont régénérés par calcul ;

Les procédures stockées

```
CREATE OR REPLACE FUNCTION
charge_remplace_ano
(table, colonne, type de données)
```

Charge toutes les valeurs distinctes qui doivent être anonymisées.

CREATE OR REPLACE FUNCTION
charge_remplace_ano
(table, colonne, type de données)

réalise simplement :

```
INSERT INTO remplace_ano (ancienne, type)
   SELECT DISTINCT(colonne_a_anonymiser),
        'type de données' FROM table
  EXCEPT
 (SELECT ancienne, type FROM remplace_ano
   WHERE type = 'type de données')';
```

Les procédures stockées

```
CREATE OR REPLACE FUNCTION
init_remplace_ano
(type de données)
```

Pour chaque valeur chargée dans la table de remplacement, associe une nouvelle valeur tirée au hasard depuis le référentiel.

Les procédures stockées

```
CREATE OR REPLACE FUNCTION
remplace_ano_direct_update
(table, colonne, type de données)
```

Applique les changements: remplace une valeur par son équivalent « *anonyme* », selon la table de remplacement.

La suite ?

- Optimisation des temps de traitement ;
- Interface utilisateur avec gestion d'un pseudo-workflow pour créer des jobs reproductibles ;
- Remplacer l'ETL Talend par DBI-Link ;
- Arrêter de mélanger des termes en français et en anglais dans mon code ;
- Ouvrir le code ? Wait & see...

Conclusion

Une utilisation sans prétention de PostgreSQL, sans utiliser de fonctionnalités avancées, simplement des procédures stockées facilement manipulables.

La partie technique est simple, le principal défi est de constituer un référentiel de données suffisamment complet.

Des questions ?

Je tiraille mon chef pour ouvrir le code ...