UGA - IUT2 - Département informatique
BUT 1A Semestre 2 - Exploitation d'une BD - 2021/2022

R2.06 et S2.04 - Contrôle du 16 Juin 2022 - Durée : 1h30

Aucun document autorisé.
Des extraits de la documentation postgreSQL sont fournis en fin d'énoncé.
Le premier exercice compte pour la SAÉ 2.04 - Exploitation d'une base de données.
Le deuxième exercice compte pour la ressource R2.06.

1. SAÉ 2.04 - Questions autour du nettoyage des données

Dans cet exercice nous vous demandons de reproduire et expliquer des opérations de nettoyage des données que vous avez réalisées pendant la SAÉ 2.04.

Nous considérons la même table **`openfoodfacts`** dont nous rappelons que les seuls champs utiles sont **parmi** les suivants :

- **`code`**
- **`url`**
- **`product_name`**
- **`brands_tags`**
- **`stores`**
- **`food_groups`**
- **`labels_tags`**
- **`countries`**
- **`countries_tags`**
- **`quantity`**
- **`fat_100g`**
- **`saturated_fat_100g`**
- **`sugars_100g`**
- **`proteins_100g`**
- **`carbohydrates_100g`**
- **`energy_100g`**
- **`salt_100g`**
- **`sodium_100g`**
- **`nutriscore_score`**
- **`nutriscore_grade`**

1. Quelle condition de sélection faut-il utiliser pour ne conserver que les produits relatifs à la Belgique? Nous rappelons que ce pays s'écrit également Belgium en anglais, Belgie en flamand, Belgien en allemand, Belgica en espagnol, Belgio en italien, ou Belgia en roumain. En revanche il ne faut pas le confondre avec le Belarus ni avec le Belize.
2. Quelle requête permet de vérifier que les lignes concernant la Belgique et la famille de produit **`'en:sweets'`**, ne contiennent pas deux fois le même code barres?
3. Nous considérons la requête suivante :

```
select product_name, food_groups, nutriscore_grade
from openfoodfacts
where stores='Penny market' and nutriscore_grade is not null
order by food_groups;
```

dont le résultat est

```
          product_name           |        food_groups         | nutriscore_grade
---------------------------------+----------------------------+------------------
 Panini per Hamburger con sesamo | en:bread                   | c
 Muesli alla frutta              | en:breakfast-cereals       | c
 Filini all'uovo                 | en:cereals                 | a
 Piombi                          | en:cereals                 | a
 Stracchino                      | en:cheese                  | d
 Filetto di salmone selvaggio    | en:fish-and-seafood        | a
 8 biscotti panna e cacao        | en:ice-cream               | d
 Premium tiramisù                | en:ice-cream               | d
 Latte microfiltrato             | en:milk-and-yogurt         | a
 Mandorle                        | en:nuts                    | a
 zuppa alla toscana              | en:one-dish-meals          | b
 Lasagne al ragù                 | en:one-dish-meals          | b
 Pizza Margherita                | en:pizza-pies-and-quiches  | c
 Latte cappuccino                | en:sweetened-beverages     | e
 Datterini gialli                | en:vegetables              | a
(15 rows)
```

Quel est le résultat de la requête suivante?

```
select distinct on (food_groups) product_name, food_groups, nutriscore_grade
from openfoodfacts
where stores='Penny market' and nutriscore_grade='d'
order by food_groups;
```

4. Pourquoi, dans le travail de nettoyage de données à réaliser dans le cadre de la SAÉ, ne faut-il pas utiliser la condition de sélection suivante : **`energy_100g between 0 and 100`**?

5. Expliquer le rôle de chacune des options de la commande d'exportation **`\copy ma_table to 'mon_resultat.csv' with (liste d'options)`**
    - **`delimiter E'\t'`**
    - **`format CSV`**
    - **`HEADER`**
    - **`ENCODING 'UTF8'`**
    - **`NULL 'NA'`**

2. R 2.06 - Administration et triggers avancés

Un groupe d'amis se réunissant régulièrement pour des jeux de rôles ont décidé de créer une base de données pour faciliter la gestion des aventures des personnages qu'ils ont imaginés. Thomas Malory a proposé d'utiliser son serveur postgreSQL, sur lequel il a déjà créé un utilisateur **`malory`** pour son usage personnel et un utilisateur **`invite`** pour un projet précédent. Pour ce projet particulier, il crée en plus l'utilisateur **`lorre`** que pourra utiliser son ami maître du jeu Chuck Lorre, et le rôle **`joueur`** où les futurs joueurs tels que Sheldon Cooper et Leonard Hofstadter, devront être inscrits. Malory a créé la base **`camelot`** et a laissé Lorre créer les tables **`chevalier`**, **`haut_fait`**, **`participe`** et **`graal`** que vous avez déjà rencontrées pour le contrôle machine du 11 avril 2022.

Nous rappelons que ces tables permettent d'enregistrer les hauts-faits des chevaliers de la Table Ronde, et en particulier de leur Quête du Graal. Le maître du jeu, Chuck Lorre, incarne le père Blaise. Chaque joueur incarne un ou des chevaliers venant d'un pays donné.

Nous rappelons également que chaque haut-fait est réalisé par une équipe constituée de chevaliers qui sont alors tous considérés comme équipiers (sans chef). Au fil des hauts-faits réalisés par les chevaliers, leur mérite augmente (le père Blaise, met à jour le mérite de chacun au nouvel an, en fonction des hauts-faits réalisés l'année précédente). Enfin, dans leur quête, les chevaliers trouvent malheureusement des graals qui se révèlent être des faux (un seul Graal peut être reconnu authentique, et c'est alors définitif).

Il s'agit à présent de terminer la construction de la base de données pour qu'elle puisse être utilisable. Les commandes suivantes illustrent son état actuel.

```
camelot=> \du
                                 List of roles
 Role name |                         Attributes                         | Member of
-----------+------------------------------------------------------------+-----------
 invite    |                                                            | {}
 joueur    | Cannot login                                               | {}
 lorre     |                                                            | {}
 malory    | Create role, Create DB                                     | {}
 postgres  | Superuser, Create role, Create DB, Replication, Bypass RLS | {}
```

```
camelot=> \l
                                      List of databases
     Name      |   Owner   | Encoding |   Collate   |    Ctype    |   Access privileges
---------------+-----------+----------+-------------+-------------+-----------------------
 camelot       | malory    | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 enterprise    | malory    | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =T/malory            +
               |           |          |             |             | invite=c/malory      +
               |           |          |             |             | malory=CTc/malory
 postgres      | postgres  | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 template0     | postgres  | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
               |           |          |             |             | postgres=CTc/postgres
 template1     | postgres  | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
               |           |          |             |             | postgres=CTc/postgres
```

```
camelot=> \d
          List of relations
 Schema |   Name    | Type  | Owner
--------+-----------+-------+--------
 public | chevalier | table | lorre
 public | graal     | table | lorre
 public | haut_fait | table | lorre
 public | participe | table | lorre
(4 rows)
```

```
camelot=> \d chevalier
                     Table "public.chevalier"
     Column     |       Type        | Collation | Nullable | Default
----------------+-------------------+-----------+----------+---------
 nom            | character varying |           | not null |
 pays_naissance | character varying |           |          |
 merite         | numeric           |           |          | 0
Indexes:
    "chevalier_pkey" PRIMARY KEY, btree (nom)
Referenced by:
    TABLE "participe" CONSTRAINT "participe_equipier_fkey" FOREIGN KEY (equipier)
        REFERENCES chevalier(nom)

camelot=> select * from chevalier limit 2;
  nom   | pays_naissance | merite
--------+----------------+--------
 arthur | Tintagel       |      0
 bohort | Gaunes         |      0
(2 rows)
```

```
camelot=> \d haut_fait
                    Table "public.haut_fait"
 Column |       Type        | Collation | Nullable |   Default
--------+-------------------+-----------+----------+--------------
 id     | numeric           |           | not null |
 lieu   | character varying |           |          |
 nature | character varying |           |          |
 jour   | date              |           |          | CURRENT_DATE
Indexes:
    "haut_fait_pkey" PRIMARY KEY, btree (id)
Referenced by:
    TABLE "participe" CONSTRAINT "participe_id_hf_fkey" FOREIGN KEY (id_hf)
        REFERENCES haut_fait(id)

camelot=> select * from haut_fait limit 2;

 id |         lieu         |        nature        |    jour
----+----------------------+----------------------+------------
  1 | Forêt de Brocéliande | lapin tueur terrassé | 2021-02-25
  2 | Orcanie              | graal trouvé         | 2021-03-20
(2 rows)
```

```
camelot=> \d participe
                  Table "public.participe"
  Column  |       Type        | Collation | Nullable | Default
----------+-------------------+-----------+----------+---------
 id_hf    | numeric           |           | not null |
 equipier | character varying |           | not null |
Indexes:
    "participe_pkey" PRIMARY KEY, btree (id_hf, equipier)
Foreign-key constraints:
    "participe_equipier_fkey" FOREIGN KEY (equipier) REFERENCES chevalier(nom)
    "participe_id_hf_fkey" FOREIGN KEY (id_hf) REFERENCES haut_fait(id)
Referenced by:
    TABLE "graal" CONSTRAINT "graal_decouvreur_id_hf_fkey" FOREIGN KEY (decouvreur, id_hf)
        REFERENCES participe(equipier, id_hf)

camelot=> select * from participe limit 2;
 id_hf | equipier
-------+----------
     1 | arthur
     1 | lancelot
(2 rows)
```

```
camelot=> \d graal
                      Table "public.graal"
   Column    |       Type        | Collation | Nullable | Default
-------------+-------------------+-----------+----------+---------
 id          | numeric           |           | not null |
 authentique | boolean           |           |          | false
 decouvreur  | character varying |           |          |
 id_hf       | numeric           |           |          |
Indexes:
    "graal_pkey" PRIMARY KEY, btree (id)
Foreign-key constraints:
    "graal_decouvreur_id_hf_fkey" FOREIGN KEY (decouvreur, id_hf)
        REFERENCES participe(equipier, id_hf)

camelot=> select * from graal limit 2;
 id | authentique | decouvreur | id_hf
----+-------------+------------+-------
  1 | f           | lancelot   |     2
  2 | f           | arthur     |     5
(2 rows)
```

2.1. Sécurisation de la base

1. Quelles commandes faut-il exécuter pour créer l'utilisateur **`leonard`** avec comme mot de passe **`'hofstadter'`** et **`sheldon`** avec comme mot de passe **`'cooper'`**, de telle sorte qu'ils appartiennent tous deux au *role* **`joueur`**?
2. L'utilisateur **`lorre`** peut-il exécuter ces commandes? Justifier. Même question pour **`malory`**.
3. Quelles commandes faut-il exécuter pour que les seuls utilisateurs pouvant accéder à la base **`camelot`** soient **`mallory`**, **`lorre`**, **`postgres`** ainsi que ceux appartenant au *role* **`joueur`**? Prenez soin de proposer les commandes les plus courtes possible pour cela. Justifier qu'elles sont effectivement suffisantes.
4. L'utilisateur **`lorre`** peut-il exécuter ces commandes? Justifier. Même question pour **`malory`**.

2.2. Droits des utilisateurs

Chaque joueur peut consulter tous les tuples des tables **`haut_fait`**, **`participe`** et **`graal`**, mais seulement les chevaliers du pays qui lui est attribué. Pour cela, une vue **`chevalier_de_tintagel`** est créée pour **`leonard`** et une vue **`chevalier_de_gaunes`** pour **`sheldon`**. Les deux vues n'affichent que le nom et le mérite des chevaliers nés, respectivement, à Tintagel ou en Gaunes. Chaque joueur peut ajouter de nouveaux chevaliers, mais toujours avec un mérite égal à 0, et nés dans le pays attribué au joueur. L'utilisateur **`lorre`** a quant à lui tous les droits sur toutes les tables de la base.

5. Créer la vue **`chevalier_de_tintagel`**. Nous supposerons que la vue **`chevalier_de_gaunes`** est créée de façon similaire.
6. Ecrire une requête que pourrait exécuter **`leonard`** pour connaître le nombre de graals trouvés par chacun de ses chevaliers, y compris par les chevaliers qui n'en ont pas encore trouvé.
7. Que faut-il créer pour qu'une insertion dans chacune de ces deux vues se traduise en une insertion dans la table **`chevalier`** avec les bonnes valeurs? Les créer pour la vue **`chevalier_de_tintagel`**. Nous supposerons que des créations similaires sont réalisées pour la vue **`chevalier_de_gaunes`**.
8. Quelles commandes faut-il alors exécuter pour accorder les droits voulus aux trois utilisateurs **`leonard`**, **`sheldon`**, et **`lorre`**? Là encore, prenez soin de proposer des commandes les plus courtes possible.
9. L'utilisateur **`lorre`** peut-il exécuter ces commandes? Justifier. Même question pour **`malory`**.
10. Quels sont les droits de **`malory`** sur les tables de la base **`camelot`**?

2.3. Triggers et fonctions pour vérifier des contraintes complexes

Nous terminons par la définition d'un trigger et de sa fonction associée pour respecter la contrainte suivante : lorsque le vrai Graal a été trouvé, son découvreur devient le plus méritant qui soit.

11. Ecrire un trigger et sa fonction associée pour que la valeur `'NaN'` (plus grande que toutes les valeurs du type `numeric`) soit affectée au mérite du découvreur du vrai Graal dans les deux cas suivants :
    - l'authentification d'un graal (qui avait été préalablement enregistré comme un faux);
    - l'enregistrement du graal authentique.

    Ceci constituant un effet de bord, il est alors nécessaire d'ajouter une notification de déclenchement.
    Remarque : on fait l'hypothèse que l'on ne changera jamais le découvreur d'un Graal authentique, mais que l'on peut changer son identifiant ou celui du haut-fait qui a permis sa découverte.

Extraits de https://www.postgresql.org/docs/13/

```
GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | TRIGGER }
    [, ...] | ALL [ PRIVILEGES ] }
    ON { [ TABLE ] table_name [, ...]
         | ALL TABLES IN SCHEMA schema_name [, ...] }
    TO role_specification [, ...] [ WITH GRANT OPTION ]

GRANT { { SELECT | INSERT | UPDATE | REFERENCES } ( column_name [, ...] )
    [, ...] | ALL [ PRIVILEGES ] ( column_name [, ...] ) }
    ON [ TABLE ] table_name [, ...]
    TO role_specification [, ...] [ WITH GRANT OPTION ]

GRANT { { CREATE | CONNECT | TEMPORARY | TEMP } [, ...] | ALL [ PRIVILEGES ] }
    ON DATABASE database_name [, ...]
    TO role_specification [, ...] [ WITH GRANT OPTION ]

GRANT role_name [, ...] TO role_specification [, ...]
    [ WITH ADMIN OPTION ]
    [ GRANTED BY role_specification ]

where role_specification can be:

    [ GROUP ] role_name
  | PUBLIC
  | CURRENT_USER
  | SESSION_USER
```

```
REVOKE [ GRANT OPTION FOR ]
    { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | TRIGGER }
    [, ...] | ALL [ PRIVILEGES ] }
    ON { [ TABLE ] table_name [, ...]
         | ALL TABLES IN SCHEMA schema_name [, ...] }
    FROM role_specification [, ...]
    [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
    { { SELECT | INSERT | UPDATE | REFERENCES } ( column_name [, ...] )
    [, ...] | ALL [ PRIVILEGES ] ( column_name [, ...] ) }
    ON [ TABLE ] table_name [, ...]
    FROM role_specification [, ...]
    [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
    { { CREATE | CONNECT | TEMPORARY | TEMP } [, ...] | ALL [ PRIVILEGES ] }
    ON DATABASE database_name [, ...]
    FROM role_specification [, ...]
    [ CASCADE | RESTRICT ]

REVOKE [ ADMIN OPTION FOR ]
    role_name [, ...] FROM role_specification [, ...]
    [ GRANTED BY role_specification ]
    [ CASCADE | RESTRICT ]

where role_specification can be:

    [ GROUP ] role_name
  | PUBLIC
  | CURRENT_USER
  | SESSION_USER
```

Example. A PL/pgSQL Trigger Function for Auditing

```
CREATE OR REPLACE FUNCTION process_emp_audit() RETURNS TRIGGER AS $emp_audit$
    BEGIN
        --
        -- Create a row in emp_audit to reflect the operation performed on emp,
        -- making use of the special variable TG_OP to work out the operation.
        --
        IF (TG_OP = 'DELETE') THEN
            INSERT INTO emp_audit SELECT 'D', now(), user, OLD.*;
        ELSIF (TG_OP = 'UPDATE') THEN
            INSERT INTO emp_audit SELECT 'U', now(), user, NEW.*;
        ELSIF (TG_OP = 'INSERT') THEN
            INSERT INTO emp_audit SELECT 'I', now(), user, NEW.*;
        END IF;
        RETURN NULL; -- result is ignored since this is an AFTER trigger
    END;
$emp_audit$ LANGUAGE plpgsql;

CREATE TRIGGER emp_audit
AFTER INSERT OR UPDATE OR DELETE ON emp
    FOR EACH ROW EXECUTE FUNCTION process_emp_audit();
```