Bases de Données - TP 3

Licence 3 d'Informatique 2021-2022

SQL : Data Definition Language et Data Modification Language

---

1 Echauffement : retour sur la base Rugby du TP 2

Commencer par analyser le fichier `tp2.sql`, qui crée et remplit les tables de la base Rugby utilisée dans le TP 2. Ce fichier est composé de plusieurs parties :

- Si certaines des tables que l'on souhaite créer existent déjà, elles sont d'abord effacées.
- Les tables sont alors créées. Observer les différents types des attributs (que signifie `serial` ?) et la manière dont les contraintes sont indiquées (clefs primaires, étrangères, contraintes de non nullité).
- Les tables sont remplies avec des valeurs.

1. Observer le remplissage de la table `matchs`. Pourquoi n'y a-t-il que quatre attributs alors que la table en a cinq ?
2. Ajouter un nouveau match dans la table matchs.
3. Essayer d'ajouter un nouveau tournoi dans la table tournois. En cas d'échec, ne pas se démoraliser et passer directement à la question suivante.
4. Après avoir observé le remplissage de la table tournois et analysé le contenu des tables `matchs_mid_seq` et `tournois_tid_seq`, réfléchir, relire éventuellement le cours et trouver comment faire pour ajouter un tournoi dans la table tournois.

Au rugby pendant les coupes du monde, un joueur peut jouer avec une équipe nationale sans avoir la nationalité de cette équipe. Un joueur peut au cours de sa carrière jouer dans des équipes nationales différentes.

5. Pour refléter ceci, créer une table `joueurs` ayant comme attributs au moins `Nom`, `Prénom`, `Date de Naissance`, `Nationalité`. Choisissez les bons types pour ces attributs. Pour choisir les noms des attributs, observer le fichier `joueur1.sql` disponible sur Moodle, qui sera utilisé après pour remplir la table. Pour définir une clef primaire, on pourrait utiliser une clef composite formée de plusieurs attributs. On préférera ici avoir un identificateur supplémentaire. Quel type est adapté ?
6. Remplir ensuite la table grâce au fichier `joueur1.sql`.
7. Comment peut-on relier cette relation avec les autres afin de pouvoir répondre à des requêtes du type "*quels joueurs ont joué dans l'équipe numéro 12 ?*". Créer la relation, les attributs et les contraintes nécessaires. Choisir le nom et les attributs de la relation de sorte qu'on puisse utiliser le fichier `joueur2.sql` pour la remplir.
8. Remplir la nouvelle table avec `joueur2.sql`.
9. Ajouter dans la nouvelle table une ligne indiquant que le premier joueur a aussi joué pour l'équipe 30. Est-ce que c'est possible ?
10. Essayer de supprimer l'équipe des All Blacks. Que se passe-t-il ?
11. Modifier les contraintes de la table matchs de sorte que lorsqu'une équipe est supprimée les matchs auxquels elle a participé sont aussi supprimés. Indication : utiliser `ALTER TABLE`, `DROP CONSTRAINT` et `ADD`.
12. Modifier également de la même façon les contraintes de la table participation et de la table créée précédemment.
13. Supprimer les joueurs Italiens.
14. Supprimer les joueurs ayant joué en Italie.

2 Travailler avec l'open data : récupération et chargement d'un jeu de données

Nous travaillerons avec les données fournis par le Syndicat des Transports en Île-de-France (STIF), au format GTFS, téléchargeables sous la forme d'un fichier ZIP depuis *ce lien*. Pour traiter ce TP, nous aurons besoin des fichiers `routes`, `stop_times`, `stops` et `trips` uniquement. Ce TP est à faire de préférence en local plutôt que sur nivose (risque de surcharge de par la taille du jeu de données). Si vous n'avez pas encore installé postgres sur votre machine personnelle, c'est donc le bon moment pour le faire.

— Récupérer ces données et étudier leur contenu. Ces données respectent un format général pour les données de transport (appelé GTFS), mais elles n'utilisent pas toutes les colonnes de ce format, car certaines sont optionnelles. Vous trouverez une description du format de données GTFS implémenté par ces données sur *ce document* (qui spécifie les colonnes effectivement utilisées et leur signification).

— Créer quatre tables au sein de PostgreSQL permettant d'héberger ces données. Ces tables doivent avoir pour nom le nom du fichier correspondant (sans le .txt), et comme noms d'attributs les noms de colonnes figurant en première ligne du fichier. Pensez à proposer un type adéquat pour chaque colonne, et à spécifier les clefs primaires, clefs étrangères et éventuelles colonnes uniques (cela aura un impact important sur la performance des requêtes que nous écrirons sur cette base plus tard dans le semestre). Remarquez qu'il convient d'importer les champs contenant des nombres hexadécimaux comme des chaînes de 6 caractères ; les horaires comme des chaînes de 8 caractères.

— À l'aide de la commande `\copy` du client en ligne de commande de PostgreSQL, charger le contenu de chacun des quatre fichiers dans la table correspondante. Cette commande utilisée sous la forme :

```
\copy nom_de_table FROM fichier.txt WITH (option, ...)
```

charge le contenu du fichier (stocké chez le client) dans la table. Les options possibles sont décrites ici `https://www.postgresql.org/docs/current/static/sql-copy.html` ; choisissez les bonnes options. Pour ce faire remarquez que les fichiers dont vous disposez sont au format CSV (Comma Separated Values) ; contiennent les noms des colonnes en première ligne ; les valeurs sur la même ligne sont séparées par une virgule ; les valeurs textuelles peuvent être entourées par des doubles guillemets (mais ne le sont pas systématiquement) ; si une valeur contient un double guillemet il est répété deux fois ; les valeurs absentes (NULL) sont représentées par une chaîne vide sans guillemets.

3 Description des données

La table `routes` décrits les lignes de transport en commun (par exemple la ligne B du RER, la ligne de bus 96, la ligne 1 du metro, la Navette Paris-Beauvais, etc.).

La table `trips` décrit les trajets des lignes (avec nom, direction etc., mais sans les arrêts). Chaque trajet est effectué par une ligne de transport en commun représentée par l'attribut `route_id` de la table.

La table `stop` décrit les arrêts de transport en commun. Certains ne sont pas des vrais arrêts, mais décrivent une zone desservie par plusieurs lignes de transports en commun. Les arrêts qui décrivent une zone d'arrêt sont ceux qui n'ont pas de `parent_station`.

En revanche un arrêt avec une station parent (attribut `parent_station` non-NULL) est un vrai arrêt d'une ligne de transport en commun ; sa station parent fait référence à la zone d'arrêt dans laquelle l'arrêt se trouve. Par exemple "DENFERT-ROCHEREAU" est la station parent de plusieurs autres arrêts (l'arrêt "DENFERT-ROCHEREAU-METRO-RER" l'arrêt de bus "DENFERT-ROCHEREAU - ARAGO", "DENFERT-ROCHEREAU - DAGUERRE", etc.). Cela indique que tous ces arrêts sont dans la zone de "DENFERT-ROCHEREAU" et sont donc proches entre eux.

Par conséquent on dira qu'un arrêt `s1` se trouve "dans la zone" de `s2` si `s2` est la station parent de `s1`. On appellera les stations parent des "zones d'arrêt".

La table `stop_times` décrit les arrêts desservis par chaque trajet, avec leurs horaires. Remarquer que seuls des vrais arrêts peuvent faire partie d'un trajet (c'est-à-dire qu'on ne retrouve pas dans cette table les zones d'arrêts, qui sont des arrêts fictifs).