

# TP SQL

Ce TP n'est pas noté. Vos enseignants ne récupéreront pas de comptes-rendus.  
Ce TP se déroulera sur 2 séances (première séance: jusque question 2.5 à peu près)

Code couleur (sans importance): en général **outil** est un nom de logiciel, **variable** un nom/variable que l'on pourrait changer, **\d** est une instruction spécifique à psql, **select \* from t;** est une instruction sql, **attr** est un nom de table ou d'attribut.

Une explication

Une remarque/instruction

Autre aide/instruction

Des instructions/résultats dans un terminal

## 1 Environnement

### 1.1 Se connecter à la base, manipuler des données

#### 1.1.1 Connection à une base

Nous allons pour ce TP utiliser le SGBD PostgreSQL.

1. Commencer par mettre en place l'environnement (un conteneur dans lequel un serveur postgresQL tourne en tâche de fond et dans lequel on utilise un client psql connecté au serveur postgresQL). Quelques détails sur notre solution:
  - Vous devez impérativement adopter un comportement responsable et utiliser la ressource uniquement pour exécuter les requêtes du TP (c'est vrai pour toute utilisation des moyens de l'université à vrai dire).
  - Stockez les données importantes (pas la BD, mais par exemple les requêtes que vous tapez ou vos notes perso) dans des fichiers en dehors du conteneur docker. Les conteneurs docker de l'université ne sont pas des ressources pérennes (contrairement à ceux utilisés ailleurs, sur votre ordinateur ou autre) parce que les ressources sont limitées et partagées entre de nombreux utilisateurs, donc des conteneurs sont régulièrement supprimés sans préavis.

En lançant le client **psql**, votre terminal ne peut désormais plus exécuter que des instructions **psql** (requêtes SQL ou instructions propres à **psql** comme **\d**) jusqu'à ce que vous exécutiez l'instruction **\q** qui quitte **psql** pour revenir au terminal du conteneur. Les invites de commande **psql** diffèrent de celle du terminal.

(Plus tard) À la fin de la séance, vous penserez à supprimer votre conteneur docker (on pourrait se contenter de l'inactiver, mais il est tout aussi simple de le supprimer et recréer un nouveau lorsque vous souhaitez poursuivre le TP dans notre cas). Cela permet de libérer les ressources.

2. Ouvrir un fichier avec un éditeur de texte (ex: gedit ou geany, voire emacs ou vi si vous connaissez. Pas word ni Google doc) sur votre machine. Appelez ce fichier **tp\_bd.sql** (ou un autre nom avec l'extension .sql). Vous écrirez dans ce fichier vos instructions SQL, pour ensuite les copier-coller dans le terminal (si vous préférez vous pouvez faire l'inverse: taper directement dans le terminal, et copier dans le fichier pour garder une trace de ce que vous avez fait; mais il est plus facile d'éditer dans le fichier). Penser à sauvegarder périodiquement ce fichier pour se rappeler ce que vous avez écrit, et à formater de façon lisible vos requêtes.

- utiliser l'extension .sql permet à l'éditeur de reconnaître que c'est du sql et donc d'utiliser la coloration syntaxique.
- commentaires en SQL: voir ci-dessous
- Quelques règles que j'impose pour formater les requêtes:
  - un commentaire avant chaque requête indiquant son numéro
  - passer à la ligne là où c'est pertinent (au moins avant chaque clause d'une requête; SELECT, FROM, WHERE,...).
- des instructions spécifiques à sql sont données dans le fichier "aide-psql.pdf".
- Le schéma de la base de données Northwind utilisée en TP est donné dans le fichier "northwind-schema.png". Ce schéma est légèrement incomplet: on en a exclu certaines tables présentes dans la base mais inutiles au TP.
- Quelques raccourcis claviers et autres commandes utiles (par exemple: pour coller dans le terminal: Ctrl+Shift+V) sont donnés dans le fichier "aide-informatique.pdf"

### 1.1.2 Création d'une table, contraintes d'intégrité, mises à jours

Nous allons commencer par quelques manipulations sur une table toute simple. Vous trouverez ci-dessous les instructions dont vous avez besoin, mais vous pouvez jeter un bref coup d'oeil à la documentation de PostgreSQL: <https://www.postgresql.org/docs/current/sql-altertable.html>

```
-- Instructions SQL pour les créer une table
CREATE TABLE nom_table(
  attribut_1 type_1,
  ...
  attribut_n type_n
);

/* et pour ajouter à une table existante une contrainte d'intégrité nommée
(vous pouvez nommer la contrainte comme vous voulez mais la convention ci-dessous est pratique): */
ALTER TABLE nom1 ADD CONSTRAINT nomt1_pkey PRIMARY KEY (attribut1, attribut2);
ALTER TABLE t2 ADD CONSTRAINT t2_ma_fkey FOREIGN KEY (a1, a2) REFERENCES nom1(attr1, attr2);
```

1. Mises à jours et contraintes d'intégrité, clés primaires.

- Créer une table `t` contenant deux attributs: `nom` de type text, et `age` de type int.
- Ajouter une contrainte à votre table, qui définit `nom` comme clé primaire de la table.
- Observer et comprendre ce qui se passe lorsque l'on insère successivement, une à une les lignes suivantes dans la table <sup>1</sup>:

```
Marie, 10
Henri, 5
Alix, 5
Marie, 4
Foy, 14
```

Vous devriez observer une erreur en insérant Marie la seconde fois. Pourquoi (comprendre le message d'erreur)?

- Vérifier le contenu de la table. Puis augmenter l'âge de Henri de 2, puis encore de 3 (donc de 5 au total). Avoir deux personnes ayant 10 ans pose-t-il problème ?<sup>2</sup>
- Effacer de la table toutes les personnes ayant entre 6 ans (inclus) et 11ans (exclus).

Vous attendiez-vous à observer un message d'erreur pour cette suite d'instructions (pourquoi) ?

<sup>1</sup>Rappel: pour insérer en SQL: [la doc postgres](#)

<sup>2</sup>Rappels: pour les mises à jour en SQL : [la doc PostgreSQL](#)

## 1.2 Chargement de la base de données du TP

1. Lister les tables présentes dans la base de données par `\d`
2. Les questions précédentes ont modifié la base de données de départ (qui était vide, en lui ajoutant des tables désormais sans intérêt). Détruire et recréer une base de données vide avec les instructions suivantes:

```
drop schema public cascade;  
create schema public;
```

- En fait on ne détruit pas vraiment toute la base de données, on ne détruit que le "schéma postgresql" utilisé par défaut, qui s'appelle "public".
  - Cette notion de schéma qui isole une partie de la base de données est propre à postgresql, mais c'est bien pratique ici pour éviter de devoir écrire 14 instructions "drop table nomtable cascade".
3. (si ce n'est déjà fait) Télécharger et copier vers le conteneur le script permettant de créer la base de données, en exécutant dans un terminal (pas dans le client postgres) les instructions données au point 4 de <https://www.lri.fr/~groz/documents/peip2-23-24/exercices.html> (dockersh, wget url, docker cp fichier conteneur:/)
  4. Exécuter le script de chargement dans le client psql

```
\i base-northwind-postgresql-copy.sql
```

5. `\i monfichier.sql` est une instruction (spécifique à `psql` comme les autres instructions commençant par `\` dont `\d`) exécute les instructions SQL et `psql` contenues dans le fichier.
6. Vérifier en listant les tables présentes dans la base reconstruite : `\d`
7. Afficher le schéma de la table employees à l'aide de l'instruction `\d employees`

Il s'agit d'une base de données d'une entreprise fictive appelée *Northwind Traders*, où on retrouve les tables typiques de ce genre de bases de données: fournisseurs (🇬🇧suppliers), clients (🇬🇧customers), commandes (🇬🇧orders)... Cette base a été utilisée par Microsoft pour présenter les fonctionnalités de son SGBD *Access*. Le script est une suite d'instructions SQL de type *INSERT*. On aurait pu charger plus efficacement les données depuis des fichiers .csv avec l'instruction *COPY* spécifique à PostgreSQL (mais qui dispose d'équivalents dans les autres SGBDs).

## 2 Requêtes SQL

En vous aidant du schéma de la base de données (dont une illustration est fournie sur la page du cours), exprimez en SQL les requêtes suivantes (l'énoncé vous a fourni en général une illustration du résultat attendu):

1. Liste des employés travaillant au Royaume-Uni (UK) avec leur nom, prénom, et ville.

```
firstname | lastname | city  
-----+-----+-----  
Steven    | Buchanan | London  
...  
(4 rows)
```

2. Liste des villes dans lesquelles travaille au moins un employé.

```
city  
-----  
London  
Tacoma  
Seattle  
Redmond  
Kirkland  
(5 rows)
```

3. Nom du (ou des) fournisseur produisant de la Chartreuse verte.

```
company_name
-----
Aux joyeux ecclésiastiques
(1 row)
```

4. Liste des compagnies avec les catégories de produits qu'elles fournissent (on n'affichera pas les compagnies ne produisant aucun produit). Les paires seront triées à l'intérieur d'une même catégorie par ordre alphabétique de compagnie, et triées plus généralement par ordre alphabétique inverse de catégorie.<sup>3</sup>

```
company_name | category_name
-----+-----
Escargots Nouveaux | Seafood
...
Aux joyeux ecclésiastiques | Beverages
...
Refrescos Americanas LTDA | Beverages
(49 rows)
```

5. Liste des compagnies fournissant des fruits de mer ou des boissons, par ordre alphabétique.

```
company_name
-----
Aux joyeux ecclésiastiques
...
(15 rows)
```

6. Liste des compagnies fournissant des fruits de mer mais pas de boissons, par ordre alphabétique.

```
company_name
-----
Escargots Nouveaux
...
(7 rows)
```

7. Liste des compagnies fournissant des fruits de mer ( Seafood) et des boissons ( Beverages), par ordre alphabétique. Vous fournirez deux requêtes différentes.

```
company_name
-----
...
(1 row)
```

8. Liste des compagnies fournissant des fruits de mer et des boissons, par ordre alphabétique. Vous afficherez aussi le prix auquel on peut obtenir cette combinaison de produits (calculé à partir du prix unitaire de chaque).

```
company_name | amount
-----+-----
Pavlova, Ltd. | 77.50
(1 row)
```

9. Nombre d'employés

```
nb_employes
-----
9
(1 row)
```

10. Afficher le nombre de villes contenant (au moins) un employé. et nombre de pays dans lesquels on trouve des employés.

```
nb_villes | nb_pays
-----+-----
5 | 2
(1 row)
```

<sup>3</sup>Si vous avez oublié comment trier; vous pouvez regarder:  la doc française ou  anglaise de PostgreSQL.

11. Afficher pour chaque pays:

- sur une première colonne: le pays,
- sur une deuxième colonne: le nombre d'employés dans le pays
- sur une troisième colonne enfin: le nombre d'employés de ce pays dont on connaît la région (c'est à dire que la valeur de la région est renseignée - c-à-d., pas vide - dans la table).

```
country | nb_employees | nb_employees_region
-----+-----+-----
USA      |          5 |          5
UK       |          4 |          0
(1 row)
```

12. Prix du produit le plus cher

```
prix
-----
263.50
(1 row)
```

13. Ajouter sur une première colonne le nom du produit le plus cher, et sur une seconde colonne le prix correspondant (vous avez le droit d'utiliser une sous requête). En cas d'ex-aequo votre requête affichera tous les produits les plus chers.

Ensuite, proposer une seconde façon de le calculer dans le cas où on sait que le produit le plus cher est unique (ou bien s'il n'est pas unique alors on acceptera n'importe quel choix parmi les réponses possibles), en utilisant la fonction **LIMIT** qui limite le résultat d'une requête à ses premiers résultats: **LIMIT 5** n'affichera par exemple que les 5 premières lignes du résultat.<sup>4</sup>

```
product_name | prix
-----+-----
...a vous de le découvrir
(1 row)
```

14. Montant total de l'ensemble des commandes (*Indice: il faut prendre en compte 3 attributs: le prix unitaire hors réduction, le pourcentage de réduction et la quantité commandée*)

```
montant
-----
1265793.0395
(1 row)
```

15. Nombre de commandes et leur montant total par pays de client, de fournisseur, et par catégorie.

```
customer_country | supplier_country | category | nb_commandes | montant
-----+-----+-----+-----+-----
Argentina      | Australia        | Confections |          1 | 104.7000
...
```

16. Nombre de clients ayant un fax.

```
nb
---
69
(1 row)
```

17. Nombre de clients ayant un fax, et nombre de clients n'ayant pas de fax, en indiquant "fax" et "pas de fax" selon le cas. Proposer 2 solutions: la première utilisant deux **SELECT**, la seconde n'en utilisant qu'un. Pour la seconde, vous aurez sans doute à utiliser un **CASE**.

```
fax | count
-----+-----
no fax |    22
fax    |    69
(2 rows)
```

<sup>4</sup>En fait vous il y a beaucoup de façons de faire même sans utiliser **LIMIT**: vous pourriez utiliser différents types de sous-requêtes, voire même une différence ensembliste...

18. Liste des clients situés en Loire atlantique (ce département français a pour numéro 44) et ayant un fax.

```
contact_name
-----
Janine Labrune
Carine Schmitt
(2 rows)
```

19. Liste des employées ayant traité au moins 50 commandes. Attention, 2 employés peuvent avoir le même nom de famille, auquel cas ils doivent être tous deux représentés. Enfin, on affichera le résultat trié par ordre alphabétique sur les employés, mais lorsque 2 employés ont le même nom on les départagera en affichant d'abord celui qui a le plus de commandes.

```
lastname | nb
-----+-----
Callahan | 104
...
Suyama   | 67
(7 rows)
```

20. Afficher pour chaque produit (par ordre alphabétique) le nom du produit et le nombre de commandes dans lesquelles le produit apparaît. Attention : on veut aussi afficher les produits qui n'ont jamais été commandés (avec un 0 pour le nombre de commandes). Contrainte supplémentaire: on vous demande de n'utiliser qu'une seule fois le mot-clé SELECT dans votre requête.

```
product_name | nb
-----+-----
Alice Mutton | 37
Aniseed Syrup | 12
...
Wimmers gute Semmelknodel | 30
Zaanse koeken | 21
(77 rows)
```