Introduction à PostgreSQL

Bases de données 2

Thibaut MADELAINE

janvier 2022

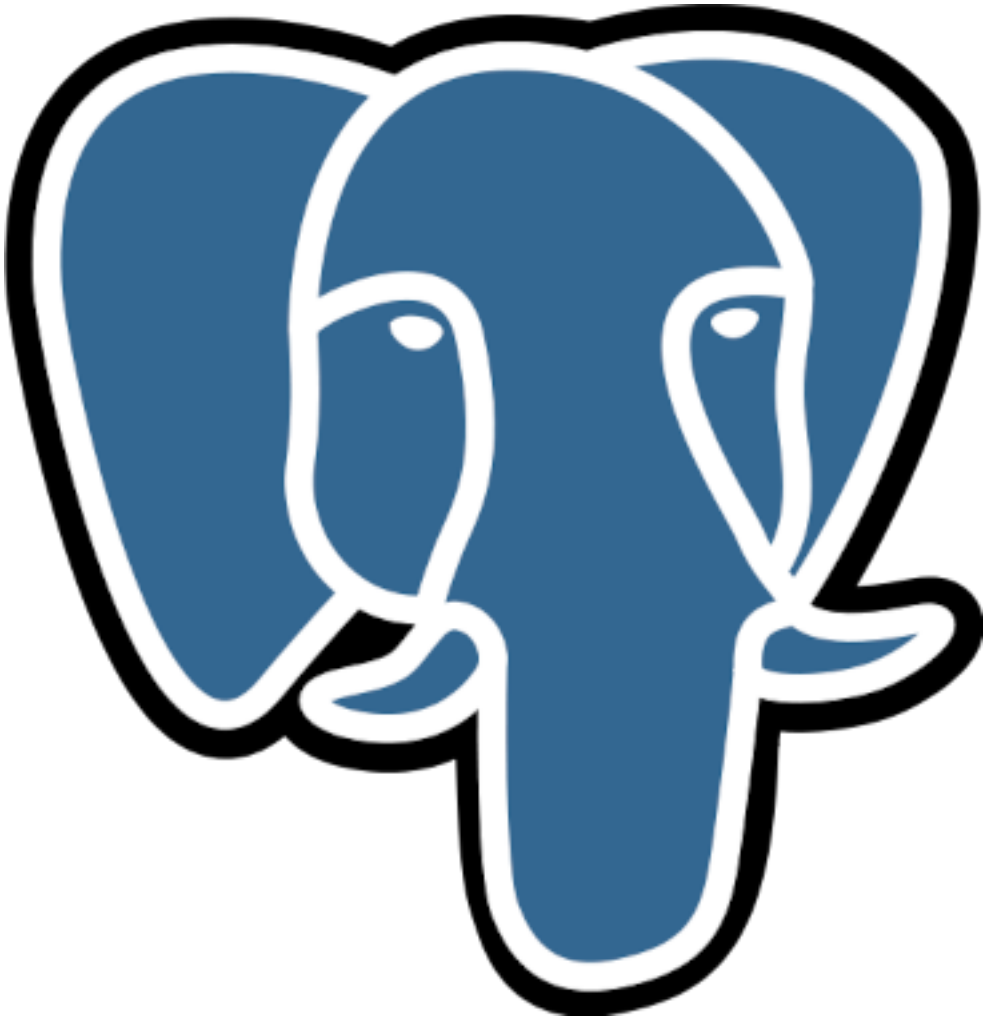
Chers lectrices & lecteurs,

Cette formation PostgreSQL est issue des manuels Dalibo. Ils ont été repris par Thibaut MADELAINE pour rentrer dans le format universitaire avec Cours Magistraux, Travaux Dirigés (sans ordinateurs) et Travaux Pratiques (avec ordinateur).

Au-delà du contenu technique en lui-même, l'intention des auteurs est de transmettre les valeurs qui animent et unissent les développeurs de PostgreSQL depuis toujours : partage, ouverture, transparence, créativité, dynamisme… Le but premier de cette formation est de vous aider à mieux exploiter toute la puissance de PostgreSQL mais nous espérons également qu'elles vous inciteront à devenir un membre actif de la communauté en partageant à votre tour le savoir-faire que vous aurez acquis avec nous.

Nous mettons un point d'honneur à maintenir nos manuels à jour, avec des informations précises et des exemples détaillés. Toutefois malgré nos efforts et nos multiples relectures, il est probable que ce document contienne des oublis, des coquilles, des imprécisions ou des erreurs. Si vous constatez un souci, n'hésitez pas à le signaler sur le site gitlab https://gitlab.com/madtibo/cours_dba_pg_universite/-/issues !

Découvrir PostgreSQL



Mon entreprise : SpaceFill

- Plateforme en ligne de stockage en entrepôts
- Développeur FullStack
- Site : https://spacefill.fr

Équipe : https://www.welcometothejungle.com/fr/companies/spacefill

---

Programme de ce cours

- **Semaine 1** : découverte de PostgreSQL
  - Histoire
  - Concepts et fonctionnalités
  - l'outil `psql`
- Semaine 2 : transactions et accès concurrents
- Semaine 3 : missions du DBA
- Semaine 4 : optimisation et indexation
- Semaine 5 : *PL/PgSQL* et triggers

Organisation du cours

- Des Cours Magistraux : à lire !
- Des Travaux Dirigés
- Des Travaux Pratiques
- Un examen final

PostgreSQL : c'est...

Le Système de Gestion de Base de Données relationnelle
Open Source le plus avancé au monde

Définitions

- Modèle relationnelle
- Base de données
- Système de gestion de base de données

Recherchons ces concepts sur Wikipedia.

Modèle relationnel

- Inventé par Edgar F. Codd en 1970
- Organisation de l'information dans des *relations* ou *tables*
  - *attributs* : les colonnes
  - *nuplets* ou *enregistrements* : les lignes
- *base de données* : regroupement de plusieurs relations

En informatique, une **base de données relationnelle** est une base de données où l'information est organisée dans des tableaux à deux dimensions appelés des *relations* ou *tables*, selon le modèle introduit par Edgar F. Codd en 1970.
Selon ce modèle relationnel, une base de données consiste en une ou plusieurs relations. Les lignes de ces relations sont appelées des *nuplets* ou *enregistrements*. Les colonnes sont appelées des *attributs*.

Une base de données…

est un *conteneur* stockant des données pouvant être retraités par des moyens informatiques pour produire une information.

Une **base de données** est un *conteneur* stockant des données pouvant être retraités par des moyens informatiques pour produire une information.

Un système de gestion de base de données…

est un logiciel système servant à
stocker, manipuler ou gérer, et à partager
des informations dans une base de données.

Un SGBD garantie :

- la qualité, la pérennité et la confidentialité des informations,
- tout en cachant la complexité des opérations.

Histoire de PostgreSQL

PostgreSQL : un logiciel Open Source

- sous licence PostgreSQL
- Droit de :
  - utiliser
  - copier
  - modifier
  - distribuer sans coût de licence

La licence PostgreSQL[1] est issue des licences BSD et MIT. Elle offre les droits de utiliser, copier, modifier, distribuer sans coût de licence. Elles est reconnu par l'Open Source Initiative[2]. Elle est utilisée par un grand nombre de projets de l'écosystème

Cette licence vous donne le droit de distribuer PostgreSQL, de l'installer, de le modifier… et même de le vendre. Certaines sociétés, comme EnterpriseDB, produisent leur version de PostgreSQL de cette façon.

PostgreSQL : quelle histoire !

- Parmi les plus vieux logiciels libres
- Et les plus sophistiqués
- Souvent cité comme exemple :
  - qualité du code

[1] http://www.postgresql.org/about/licence/
[2] http://opensource.org/licenses/PostgreSQL

  - indépendance des développeurs
  - réactivité de la communauté

L'histoire de PostgreSQL est longue, riche et passionnante. Au côté des projets libres Apache et Linux, PostgreSQL est l'un des plus vieux logiciels libres en activité et fait partie des SGBD les plus sophistiqués à l'heure actuelle.

Au sein des différentes communautés libres, PostgreSQL est souvent utilisé comme exemple à différents niveaux :

- qualité du code ;
- indépendance des développeurs et gouvernance du projet ;
- réactivité de la communauté ;
- stabilité et puissance du logiciel.

Tous ces atouts font que PostgreSQL est désormais reconnu et adopté par des milliers de grandes sociétés de par le monde.

¿¡¿¡ PostgreSQL ?!?!

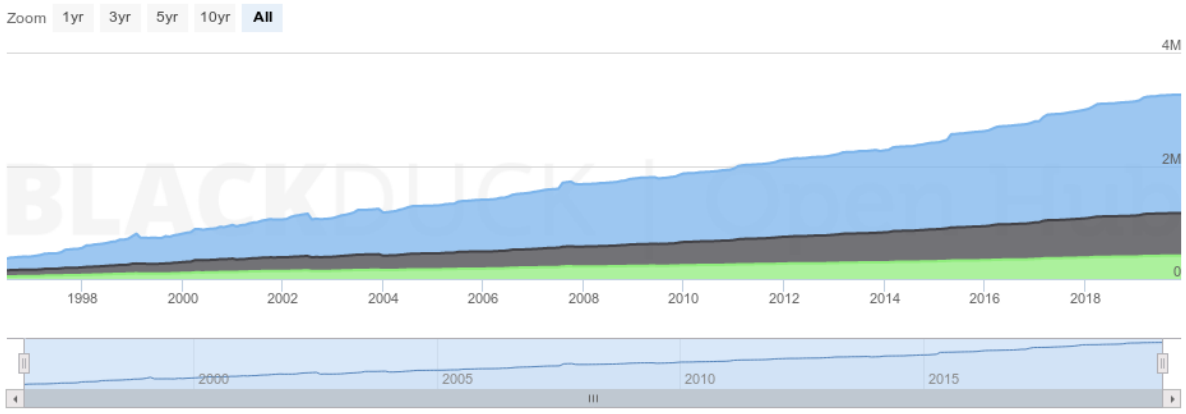
- Michael Stonebraker recode Ingres
- post « ingres » => post~~in~~gres => postgres
- support de SQL : postgre => PostgreSQL

L'origine du nom PostgreSQL remonte à la base de données Ingres, développée à l'université de Berkeley par Michael Stonebraker. En 1985, il prend la décision de reprendre le développement à partir de zéro et nomme ce nouveau logiciel Postgres, comme raccourci de post-Ingres.

En 1995, avec l'ajout du support du langage SQL, Postgres fut renommé Postgres95 puis PostgreSQL.

Aujourd'hui, le nom officiel est « PostgreSQL » (prononcez « post - gresse - Q - L »). Cependant, le nom « Postgres » est accepté comme alias.

Progression du code



Ce graphe représente l'évolution du nombre de lignes de code dans les sources de PostgreSQL. Cela permet de bien visualiser l'évolution du projet en terme de développement.

On note une augmentation constante depuis 1998 avec une croissance régulière d'environ 100 000 lignes de code C par an. Le plus intéressant est certainement de noter que l'évolution est constante.

Actuellement, PostgreSQL est composé de plus de 1 766 134 de lignes (dont 425 730 lignes de commentaires), pour environ 200 développeurs actifs[3].

Source[4].

Concepts de base

- ACID
- MVCC
- Transactions
- WAL (journaux de transactions)
- Politique de versionnage

---

[3] https://www.postgresql.org/community/contributors/
[4] https://www.openhub.net/p/postgres

ACID

- **Atomicité** (Atomic)
- **Cohérence** (Consistent)
- **Isolation** (Isolated)
- **Durabilité** (Durable)

Les propriétés ACID sont le fondement même de tout système transactionnel. Il s'agit de quatre règles fondamentales :

- A : Une transaction est entière : « tout ou rien ».
- C : Une transaction amène le système d'un état stable à un autre.
- I : Les transactions n'agissent pas les unes sur les autres.
- D : Une transaction validée provoque des changements permanents.

Les propriétés ACID sont quatre propriétés essentielles d'un sous-système de traitement de transactions d'un système de gestion de base de données. Certains SGBD ne fournissent pas les garanties ACID. C'est le cas de la plupart des SGBD non-relationnels (« NoSQL »). Cependant, la plupart des applications ont besoin de telles garanties et la décision d'utiliser un système ne garantissant pas ces propriétés ne doit pas être prise à la légère.

MultiVersion Concurrency Control (MVCC)

- Le « noyau » de PostgreSQL
- Garantit ACID
- Permet les écritures concurrentes sur la même table

MVCC (Multi Version Concurrency Control) est le mécanisme interne de PostgreSQL utilisé pour garantir la cohérence des données lorsque plusieurs processus accèdent simultanément à la même table.

C'est notamment MVCC qui permet de sauvegarder facilement une base *à chaud* et d'obtenir une sauvegarde cohérente alors même que plusieurs utilisateurs sont potentiellement en train de modifier des données dans la base.

C'est la qualité de l'implémentation de ce système qui fait de PostgreSQL un des meilleurs SGBD au monde : chaque transaction travaille dans son image de la base, cohérent du début à la fin de ses opérations. Par ailleurs les écrivains ne bloquent pas les lecteurs et les lecteurs ne bloquent pas les écrivains, contrairement aux SGBD s'appuyant sur des verrous de lignes. Cela assure de meilleures performances, un fonctionnement plus fluide des outils s'appuyant sur PostgreSQL.

Transactions

- Intimement liées à ACID et MVCC :
  - Une transaction est un ensemble d'opérations atomique
  - Le résultat d'une transaction est « tout ou rien »
- mots clés `BEGIN`, `COMMIT` et `ROLLBACK`

`BEGIN` initie une transaction. `COMMIT` valide toutes les commandes depuis l'ordre `BEGIN`. `ROLLBACK` annule toutes les commandes depuis l'ordre `BEGIN`.

Write Ahead Logs, aka WAL

- Chaque donnée est écrite **2 fois** sur le disque !
- Sécurité quasiment infaillible
- Comparable à la journalisation des systèmes de fichiers

Les journaux de transactions (appelés parfois WAL ou XLOG) sont une garantie contre les pertes de données.

Il s'agit d'une technique standard de journalisation appliquée à toutes les transactions.

Ainsi lors d'une modification de donnée, l'écriture au niveau du disque se fait en deux temps :

- écriture immédiate dans le journal de transactions ;
- écriture à l'emplacement final lors du prochain `CHECKPOINT`.

Ainsi en cas de crash :

- PostgreSQL redémarre ;
- PostgreSQL vérifie s'il reste des données non intégrées aux fichiers de données dans les journaux (mode recovery) ;
- si c'est le cas, ces données sont recopiées dans les fichiers de données afin de retrouver un état stable et cohérent.

Plus d'information : https://public.dalibo.com/archives/publications/glmf108_postgresql_et_ses_journaux_de_transactions.pdf

---

Avantages des WAL

- Un seul *sync* sur le fichier de transactions
- Le fichier de transactions est écrit de manière séquentielle
- Les fichiers de données sont écrits de façon asynchrone
- Point In Time Recovery
- Réplication (*WAL shipping*)

Les écritures se font de façon séquentielle, donc sans grand déplacement de la tête d'écriture. Généralement, le déplacement des têtes d'un disque est l'opération la plus coûteuse. L'éviter est un énorme avantage.

De plus, comme on n'écrit que dans un seul fichier de transactions, la synchronisation sur disque peut se faire sur ce seul fichier, à condition que le système de fichiers le supporte.

L'écriture asynchrone dans les fichiers de données permet là-aussi de gagner du temps.

Mais les performances ne sont pas la seule raison des journaux de transactions. Ces journaux ont aussi permis l'apparition de nouvelles fonctionnalités très intéressantes, comme le PITR et la réplication physique.

Politique de versionnage

- Depuis la version 10 : X.Y
- Avant la version 10 : X.Y.Z

- Depuis la version 10
  - X : version majeure (11, 14)
  - X.Y : version mineure (13.5)
- Avant la version 10
  - X.Y : version majeure (8.4, 9.6)
  - X.Y.Z : version mineure (9.5.8)

Versions majeures

- 5 versions majeures supportées

- une nouvelle version majeure par an
  - apporte des nouvelles fonctionnalités
- actuellement : 10, 11, 12, 13 et 14
- prochaine version : 15

La philosophie générale des développeurs de PostgreSQL peut se résumer ainsi :

> *Notre politique se base sur la qualité, pas sur les dates de sortie.*

Toutefois, même si cette philosophie reste très présente parmi les développeurs, depuis quelques années, les choses évoluent et la tendance actuelle est de livrer une version stable majeure tous les 12 à 15 mois, tout en conservant la qualité des versions. De ce fait, toute fonctionnalité supposée pas suffisamment stable est repoussée à la version suivante.

La tendance actuelle est de garantir un support pour chaque version courante pendant une durée minimale de 5 ans.

La version 9.5 n'est plus supportée depuis novembre 2021, la 9.6 depuis novembre 2021. La prochaine version qui subira ce sort est la 10, en février 2022. Le support de la version 14 devrait durer jusque 2026.

Pour plus de détails : http://www.postgresql.org/support/versioning/

Versions mineures

- au moins une mise à jour mineure par trimestre
- … et en cas de faille de sécurité
- pour toutes les versions majeures supportées
- pas de nouvelle fonctionnalité

Une version mineure ne comporte que des corrections de bugs ou de failles de sécurité. Elles sont plus fréquentes que les versions majeures, avec un rythme de sortie trimestriel, sauf bug majeur ou faille de sécurité. Chaque bug est corrigé dans toutes les versions stables actuellement maintenues par le projet.

Dernières mise à jour le 12 novembre 2021 : * version 9.6.24 * version 10.19 * version 11.14 * version 12.9 * version 13.5 * version 14.1

Prochaine mise à jour prévue le 10 février 2022.

Pour plus de détails : http://www.postgresql.org/developer/roadmap

---

Compatibilité

- mise à jour majeure : migration des données
- mise à jour mineure : redémarrage

Une version majeure apporte de nouvelles fonctionnalités, des changements de comportement, etc. Le format de stockages des données internes est souvent modifié. Le passage d'une version majeure à une nouvelle version majeure nécessitera une migration des données.

Les mises à jour mineures se font en général sans soucis, et ne nécessitent en général qu'un redémarrage. Mais comme pour toute mise à jour, il faut être prudent sur d'éventuels effets de bord. En particulier on lira les *Release Notes* et si possible on testera ailleurs qu'en production.

Structuration de PostgreSQL

- Organisation physique
- Organisation logique

Organisation physique

- stockage dans le répertoire de données
- souvent appelé PGDATA
- localisations des fichiers différentes suivant les OS

Le répertoire de données est souvent appelé PGDATA, du nom de la variable d'environnement que l'on peut faire pointer vers lui pour simplifier l'utilisation de nombreux utilitaires PostgreSQL. On peut aussi le connaître, étant connecté à la base, en interrogeant le paramètre `data_directory`.

```
postgres=# SHOW data_directory ;
      data_directory
----------------------------
 /var/lib/postgresql/14/main
(1 row)
```

En version 14 :

```
postgres$ ls $PGDATA
# ls $PGDATA
base          pg_notify     pg_subtrans  postgresql.auto.conf
global        pg_replslot   pg_tblspc    postmaster.opts
pg_commit_ts  pg_serial     pg_twophase  postmaster.pid
pg_dynshmem   pg_snapshots  PG_VERSION
pg_logical    pg_stat       pg_wal
pg_multixact  pg_stat_tmp   pg_xact
```

Vous pouvez trouver une description de tous les fichiers et répertoires dans la documentation officielle[5].

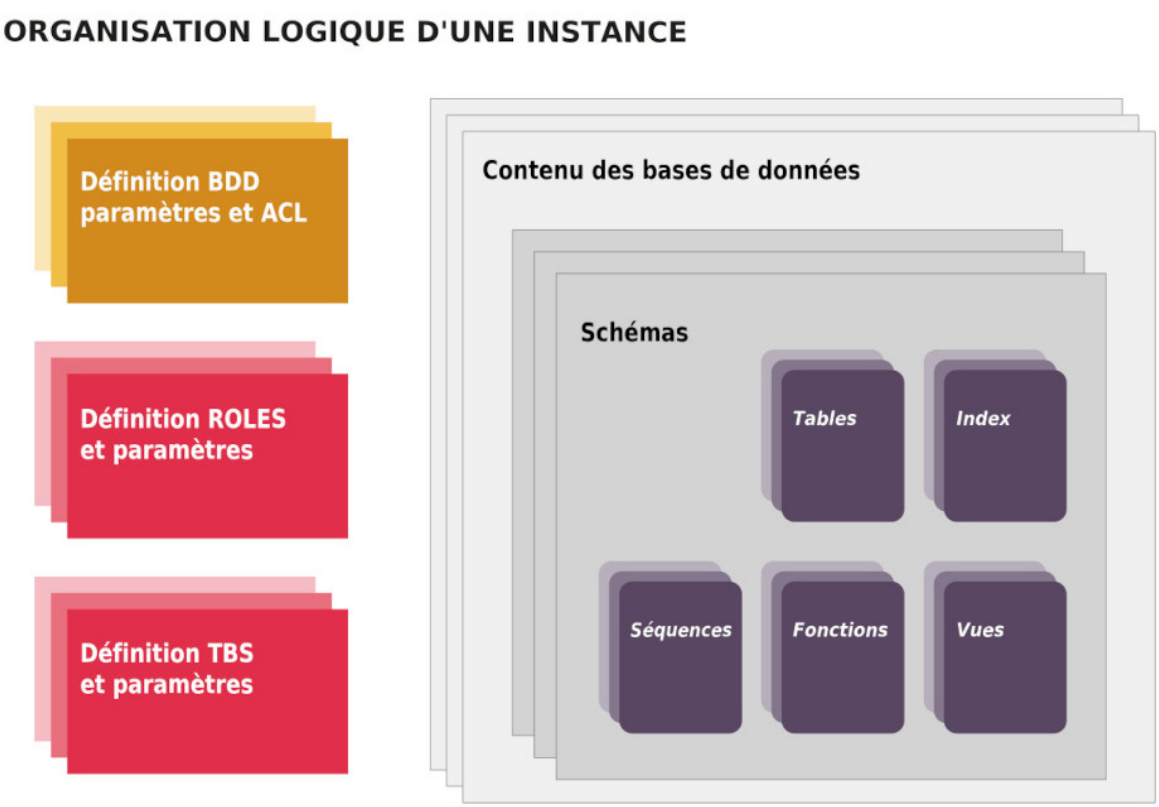
Sur les système d'exploitation de type RedHat et CentOS, le répertoire des données est situé par défaut dans `/var/lib/pgsql`.

Sur les système d'exploitation de type Debian et Ubuntu, le répertoire des données est situé par défaut dans `/var/lib/postgresql`, les fichiers de configuration est situé dans `/etc/postgresql` et le répertoire de log applicatif est situé dans `/var/log/postgresql`.

---

[5] https://www.postgresql.org/docs/current/static/storage-file-layout.html

Organisation logique



Schémas

- Espace de noms
- Concept différent des schémas d'Oracle
- Sous-ensemble de la base

Un utilisateur peut avoir accès à tous les schémas ou à un sous-ensemble. Tout dépend des droits dont il dispose. PostgreSQL vérifie la présence des objets par rapport au paramètre `search_path` valable pour la session en cours lorsque le schéma n'est pas indiqué explicitement pour les objets d'une requête.

À la création d'un utilisateur, un schéma n'est pas forcément associé.

Le comportement et l'utilisation des schémas diffèrent donc d'avec Oracle.

Les schémas sont des espaces de noms dans une base de données permettant :

- de grouper les objets d'une base de données ;
- de séparer les utilisateurs entre eux ;
- de contrôler plus efficacement les accès aux données ;
- d'éviter les conflits de noms dans les grosses bases de données.

Les schémas sont très utiles pour les systèmes de réplication (Slony, bucardo).

Exemple d'utilisation de schéma :

```
=> CREATE SCHEMA pirates;
CREATE SCHEMA
=> SET search_path TO pirates,public;
SET
=> CREATE TABLE capitaines (id serial, nom text);
CREATE TABLE

=> INSERT INTO capitaines (nom)
 VALUES ('Anne Bonny'), ('Francis Drake');
INSERT 0 2

-- Sélections des capitaines... pirates
=> SELECT * FROM capitaines;
 id |      nom
----+---------------
  1 | Anne Bonny
  2 | Francis Drake
(2 rows)

=> CREATE SCHEMA corsaires;
CREATE SCHEMA
=> SET search_path TO corsaires,pirates,public;
SET
=> CREATE TABLE capitaines (id serial, nom text);
=> INSERT INTO corsaires.capitaines (nom)
 VALUES ('Robert Surcouf'), ('Francis Drake');
INSERT 0 2
-- Sélections des capitaines... français
=> SELECT * FROM capitaines;
 id |         nom
----+---------------------
  1 | Robert Surcouf
  2 | Francis Drake
```

```
(2 rows)


-- Quels capitaine portant un nom identique
-- fut à la fois pirate et corsaire ?
=> SELECT pirates.capitaines.nom
  FROM pirates.capitaines,corsaires.capitaines
  WHERE pirates.capitaines.nom=corsaires.capitaines.nom;
      nom
---------------
 Francis Drake
(1 row)
```

Intégration logicielle de PostgreSQL

- PostgreSQL est une plate-forme de développement !
- 15 langages de procédures stockées
- Interfaces natives pour ODBC, JDBC, C, PHP, Perl, etc.
- API ouverte et documentée
- Un nouveau langage peut être ajouté **sans recompilation** de PostgreSQL

Voici la liste non exhaustive des langages procéduraux supportés :

- pl/pgsql
- pl/sql
- pl/perl
- pl/python
- pl/tcl
- pl/sh
- pl/R
- pl/java
- pl/js
- pl/lolcode
- pl/scheme
- pl/php
- pl/ruby
- pl/j
- pl/lua

- pl/pgpsm
- pl/v8

PostgreSQL peut donc être utilisé comme un serveur d'applications ! Vous pouvez ainsi placer votre code au plus près des données.

Chaque langage a ses avantages et inconvénients. Par exemple, PL/pgsql est très simple à apprendre mais n'est pas performant quand il s'agit de traiter des chaînes de caractères. Pour ce traitement, il serait préférable d'utiliser PL/perl, voire PL/python. Évidemment, une procédure en C aura les meilleures performances mais sera beaucoup moins facile à coder et à maintenir. Par ailleurs, les procédures peuvent s'appeler les unes les autres quel que soit le langage.

Les applications externes peuvent accéder aux données du serveur PostgreSQL grâce à des connecteurs. Ils peuvent passer par l'interface native, la **libpq** . C'est le cas du connecteur PHP et du connecteur Perl par exemple. Ils peuvent aussi ré-implémenter cette interface, ce que fait le pilote ODBC (psqlodbc) ou le driver JDBC.

PostgreSQL à l'heure du NoSQL

- 4 technologies majeures dans le monde NoSQL
  - Stockage clé->valeur (Redis, Apache Cassandra, Riak, MongoDB)
  - Stockage documents (Apache CouchDB, MongoDB)
  - Stockage colonnes (Apache Hbase, Google BigTable)
  - Stockage graphes (Neo4j)

- PostgreSQL est-il déprécié ?

Les SGBD relationnels créés dans les années 1970 se sont progressivement imposés jusqu'à devenir le paradigme de bases de données très largement dominant au début des années 1990.

C'est dans le courant des années 2000 avec le développement de grandes entreprises internet (Google, Amazon, eBay…) brassant des quantités énormes de données et le développement de l'informatique en grappes que la domination sans partage du modèle relationnel a été remise en question car souffrant de limites rédhibitoires pour ces nouvelles pratiques : l'extensibilité.

Les performances restent bonnes avec la montée en charge en multipliant simplement le nombre de serveurs, solution raisonnable avec la baisse des coûts, en particulier si les revenus croissent en même temps que l'activité. Les systèmes géants sont les premiers concernés : énormes quantités de

données, structuration relationnelle faible (ou de moindre importance que la capacité d'accès très rapide, quitte à multiplier les serveurs).

(texte issu de l'article Wikipedia NoSQL[6])

PostgreSQL vs. NoSQL

- PostgreSQL réunit le monde relationnel et le monde NoSQL
  - Stockage clé->valeur : `hstore`
  - Stockage documents :
    - `xml`, `json` et `jsonb` (plus performant que MongoDB)
    - procédure stockée en Javascript : PL/V8
  - Stockage colonnes : `cstore_fdw`
  - Stockage graphes : agensgraph (fork de PostgreSQL)

PostgreSQL n'est pas en reste vis à vis des bases de données NoSQL. PostgreSQL permet de stocker des données au format clé->valeur. Couplé aux index GiST et GIN, ce mode de stockage s'avère comparable à MongoDB (https://wiki.postgresql.org/images/b/b4/Pg-as-nosql-pgday-fosdem-2013.pdf) ou à Redis.

PostgreSQL peut stocker des données au format JSON (depuis la version 9.2, avec de nombreuses fonctions ajoutées en 9.3). L'avantage est que les données seront validées : si une donnée n'est pas conforme au format JSON, elle sera rejetée. Avec le type natif `jsonb` (en 9.4) il est possible d'indexer le JSON, et les performances de PostgreSQL avec ce nouveau type de stockage de documents sont très supérieures à MongoDB (http://blogs.enterprisedb.com/2014/09/24/postgres-outperforms-mongodb-and-ushers-in-new-developer-reality/).

Couplé au langage PL/V8, le type de données JSON permet de traiter efficacement ce type de données. PL/V8 permet d'écrire des fonctions en langage Javascript, elles seront exécutées avec l'interpréteur V8 écrit par Google. Ces fonctions sont bien entendus indexables par PostgreSQL si elles respectent un certain nombre de pré-requis.

Le stockage colonne pour PostgreSQL consiste actuellement en une extension *Foreign Data Wrapper* nommée `cstore_fdw`, elle est développée par le société CitusData (http://www.citusdata.com/blog/76-postgresql-columnar-store-for-analytics).

Malgré cela, PostgreSQL conserve les fonctionnalités qui ont fait sa réputation et que les moteurs NoSQL ont dû abandonner :

[6] https://fr.wikipedia.org/wiki/NoSQL

- un langage de requête puissant ;
- un optimiseur de requêtes sophistiqué ;
- la normalisation des données ;
- les jointures ;
- l'intégrité référentielle ;
- la durabilité.

Un fork de PostgreSQL, agensgraph permet d'effectuer du stockage graph.

Voici enfin un lien vers une excellente présentation sur les différences entre PostgreSQL et les solutions NoSQL : http://fr.slideshare.net/EnterpriseDB/the-nosql-way-in-postgres

ainsi que l'avis de Bruce Momjian sur le choix du NoSQL pour de nouveaux projets[7]

Sponsors & Références

- Sponsors
- Références
  - françaises
  - et internationales

Au delà de ses qualités, PostgreSQL suscite toujours les mêmes questions récurrentes :

- qui finance les développements ? (et pourquoi ?)
- qui utilise PostgreSQL ?

Sponsors

- Sociétés se consacrant à PostgreSQL :
  - Crunchy Data (Tom Lane, Stephen Frost, Joe Conway, Greg Smith)
  - EnterpriseDB (Bruce Momjian, Dave Page, Simon Riggs, …)
  - PostgresPro (Russie)
  - Cybertec (Autriche), Dalibo (France), Redpill Linpro (Suède)

[7] https://momjian.us/main/blogs/pgblog/2017.html#June_12_2017

- Société vendant un fork ou une extension :
  - Citusdata, Pivotal (Heikki Linnakangas)
- Autres sociétés :
  - VMWare, Rackspace, Heroku, Conova, Red Hat
  - NTT (*streaming Replication*), Fujitsu, NEC

La liste des sponsors de PostgreSQL contribuant activement au développement figure sur le site officiel[8]. Ce qui suit n'est qu'un aperçu.

EnterpriseDB est une société américaine qui a décidé de fournir une version de PostgreSQL propriétaire fournissant une couche de compatibilité avec Oracle. Ils emploient plusieurs codeurs importants du projet PostgreSQL (dont deux font partie de la *Core Team*), et reversent un certain nombre de leurs travaux au sein du moteur communautaire. Ils ont aussi un poids financier qui leur permet de sponsoriser la majorité des grands événements autour de PostgreSQL : PGEast et PGWest aux États-Unis, PGDay en Europe.

2nd Quadrant est une société anglaise fondée par Simon Riggs, développeur PostgreSQL de longue date. Elle développe de nombreux outils autour de PostgreSQL comme pglogical, des versions dérivées comme Postgres-XL ou BDR, dont le code se retrouve souvent dans la version communautaire après maturation, ou des outils annexes comme barman ou repmgr.

Crunchy Data offre sa propre version certifiée et finance de nombreux développements.

De nombreuses autres sociétés dédiées à PostgreSQL existent dans de nombreux pays. Parmi les sponsors officiels, on peut compter Cybertec en Autrichex ou Redpill Linpro en Suède. En Russie, PostgresPro maintient une version locale et reverse aussi ses contributions à la communauté.

En Europe francophone, Dalibo participe pleinement à la communauté. La société est Major Sponsor du projet PostgreSQL[9], ce qui indique un support de longue date. Elle développe et maintient plusieurs outils plébiscités par la communauté, comme par exemple Temboard ou postgresql_anonymizer, avec de nombreux autres projets en cours, et une participation active au développement de patchs pour PostgreSQL. Elle sponsorise également des événements comme les PGDay français et européens, ainsi que la communauté francophone. Plus d'informations[10].

Des sociétés comme Citusdata et Pivotal proposent ou ont proposé leur version dérivée mais « jouent le jeu » et participent au développement de la version communautaire, notamment en cherchant à ce que leur produit n'en diverge pas.

[8] https://www.postgresql.org/about/sponsors/
[9] https://www.postgresql.org/about/sponsors/
[10] https://www.dalibo.org/contributions

Ont également contribué à PostgreSQL nombre de sociétés non centrées autour des bases de données.

Entre 2006 et 2016, le système d'exploitation Unix Solaris 10 de Sun embarquait PostgreSQL dans sa distribution de base, comme base de données de référence. Cela a pris fin avec le rachat par Oracle, sans que cela ait représenté un danger pour PostgreSQL.

NTT a financé de nombreux patchs pour PostgreSQL, notamment liés à la réplication et inclus dans la version de la communauté depuis la 9.0[11].

Fujitsu a participé à de nombreux développements aux débuts de PostgreSQL.

VMWare a longtemps employé le développeur finlandais Heikki Linnakangas. Celui-ci travaille à présent pour Pivotal mais VMWare peut compter sur Michael Paquier.

Red Hat a longtemps employé Tom Lane à plein temps pour travailler sur PostgreSQL. Il a pu dédier une très grande partie de son temps de travail à ce projet, bien qu'il ait eu d'autres affectations au sein de Red Hat. Tom Lane a travaillé également chez SalesForce, ensuite il a rejoint Crunchy Data Solutions fin 2015.

Skype a offert un certain nombre d'outils très intéressants : pgBouncer (pooler de connexion), Londiste (réplication par trigger), etc. Ce sont des outils utilisés en interne et publiés sous licence BSD comme retour à la communauté. Le rachat par Microsoft n'a pas affecté le développement de ces outils.

Des sociétés liées au cloud comme Conova (Autriche), Heroku ou Rackspace (États-Unis) figurent aussi parmi les sponsors.

Références

- Météo France
- IGN
- RATP, SNCF, Autolib
- CNAF
- MAIF, MSA
- Le Bon Coin
- Doctolib
- Air France-KLM
- Société Générale

[11] https://wiki.postgresql.org/wiki/Streaming_Replication

- Carrefour, Leclerc, Leroy Merlin
- Instagram, Zalando, TripAdvisor
- Yandex
- …et plein d'autres

Météo France utilise PostgreSQL depuis plus d'une décennie pour l'essentiel de ses bases, dont des instances critiques de plusieurs téraoctets (témoignage sur postgresql.fr[12]).

L'IGN utilise PostGIS et PostgreSQL depuis 2006[13].

La RATP a fait ce choix depuis 2007 également[14].

La Caisse Nationale d'Allocations Familiales a remplacé ses mainframes par des instances PostgreSQL[15] dès 2010 (4 To et 1 milliard de requêtes par jour).

Instagram utilise PostgreSQL depuis le début[16].

Zalando a décrit plusieurs fois son infrastructure PostgreSQL[17] et annonçait en 2018[18] utiliser pas moins de 300 bases de données en interne et 650 instances dans un cloud AWS. Zalando contribue à la communauté, notamment par son outil de haute disponibilité patroni[19].

Le DBA de TripAdvisor témoigne de leur utilisation de PostgreSQL dans l'interview suivante[20].

Dès 2009, Leroy Merlin migrait vers PostgreSQL des milliers de logiciels de caisse[21].

Yandex, équivalent russe de Google a décrit en 2016 la migration des 300 To de données de Yandex.Mail depuis Oracle vers PostgreSQL[22].

La Société Générale a publié son outil de migration d'Oracle à PostgreSQL[23].

Autolib à Paris utilisait PostgreSQL. Le logiciel est encore utilisé dans les autres villes où le service continue. Ils ont décrit leur infrastructure au PG Day 2018 à Marseille[24].

De nombreuses autres sociétés participent au Groupe de Travail Inter-Entreprises de PostgresSQLFr[25] : Air France, Carrefour, Leclerc, le CNES, la MSA, la MAIF, PeopleDoc, EDF…

[12] https://www.postgresql.fr/temoignages/meteo_france
[13] http://postgis.refractions.net/documentation/casestudies/ign/
[14] https://www.journaldunet.com/solutions/dsi/1013631-la-ratp-integre-postgresql-a-son-systeme-d-information/
[15] https://www.silicon.fr/cnaf-debarrasse-mainframes-149897.html?inf_by=5bc488a1671db858728b4c35
[16] https://media.postgresql.org/sfpug/instagram_sfpug.pdf
[17] http://gotocon.com/dl/goto-berlin-2013/slides/HenningJacobs_and_ValentineGogichashvili_WhyZalandoTrustsInPostgreSQL.pdf
[18] https://www.postgresql.eu/events/pgconfeu2018/schedule/session/2135-highway-to-hell-or-stairway-to-cloud/
[19] https://jobs.zalando.com/tech/blog/zalandos-patroni-a-template-for-high-availability-postgresql/
[20] https://www.citusdata.com/blog/25-terry/285-matthew-kelly-tripadvisor-talks-about-pgconf-silicon-valley
[21] https://wiki.postgresql.org/images/6/63/Adeo_PGDay.pdf
[22] https://www.pgcon.org/2016/schedule/attachments/426_2016.05.19%20Yandex.Mail%20success%20story.pdf
[23] https://github.com/societe-generale/code2pg
[24] https://www.youtube.com/watch?v=vd8B7B-Zca8
[25] https://www.postgresql.fr/entreprises/accueil

Cette liste ne comprend pas les innombrables sociétés qui n'ont pas communiqué sur le sujet. PostgreSQL étant un logiciel libre, il n'existe nulle part de dénombrement des instances actives.

Le Bon Coin

- Site de petites annonces
- 4è site le plus consulté en France (2017)
- 27 millions d'annonces en ligne, 800 000 nouvelles chaque jour
- Instance PostgreSQL principale : 3 To de volume, 3 To de RAM
- 20 serveurs secondaires

PostgreSQL tient la charge sur de grosses bases de données et des serveurs de grande taille.

Le Bon Coin privilégie des serveurs physiques dans ses propres datacenters.

Pour plus de détails et l'évolution de la configuration, voir les témoignages de ses directeurs technique[26] (témoignage de juin 2012) et infrastructure[27] (juin 2017), ou la conférence de son DBA Flavio Gurgel au pgDay Paris 2019[28].

Ce dernier s'appuie sur les outils classiques fournis par la communauté : pg_dump (pour archivage, car ses exports peuvent être facilement restaurés), barman, pg_upgrade.

Doctolib

- Site de rendez-vous médicaux en ligne
- Base transactionnelle de 3 To (2018)
  - SSD chiffrés, full NVMe, RAID 50
- 25 millions de visites/mois
- pointes à 40 000 commits/s
- 6 serveurs répartis sur 2 sites avec réplication en *streaming*
- Primaire avec 4x18 cœurs (144 *threads*), 512 Go de RAM
- Scaling horizontal sur un seul secondaire

[26] https://www.postgresql.fr/temoignages:le_bon_coin
[27] https://www.kissmyfrogs.com/jean-louis-bergamo-leboncoin-ce-qui-a-ete-fait-maison-est-ultra-performant/
[28] https://www.postgresql.eu/events/pgdayparis2019/schedule/session/2376-large-databases-lots-of-servers-on-premises-on-the-cloud-get-them-all/

Doctolib est le site de rendez-vous médicaux en ligne dominant sur le marché français, couvrant 65 000 professionnels de santé, 1300 cliniques, hôpitaux et centres de santé, et des millions de patients.

Une seule instance primaire PostgreSQL suffit, sans partitionnement, épaulée par une instance secondaire qui assure une fraction du trafic web, et 4 autres secondaires pour d'autres tâches.

Le RAID50 est un compromis entre le RAID 5 (pénalisant pour les écritures à cause du calcul de parité) et le RAID 10 (trop consommateur d'espace disque).

Zalando Site de ventes par correspondances :

- Bases transactionnelles de plus de 5 To
- 90 instances principales
- 800 serveurs Tomcat
- Serveurs HP G8
- Utilisation du partitionnement

Présentation[29]

Bibliographie

- Documentation officielle (préface)
- Articles fondateurs de M. Stonebracker
- Présentation du projet PostgreSQL

« Préface : 2. Bref historique de PostgreSQL[30] ». PGDG, 2013

« The POSTGRES™ data model[31] ». Rowe and Stonebraker, 1987

« Présentation du projet PostgreSQL[32] ». Guillaume Lelarge, 2008

Iconographie :

La photo initiale est le logo officiel de PostgreSQL[33].

---

[29] https://www.slideshare.net/try_except_/goto-2013whyzalandotrustsinpostgre-sql20131018
[30] http://docs.postgresqlfr.org/9.4/history.html
[31] http://db.cs.berkeley.edu/papers/ERL-M85-95.pdf
[32] http://www.dalibo.org/presentation_du_projet_postgresql
[33] http://wiki.postgresql.org/wiki/Trademark_Policy

Serveurs

- Site officiel : http://www.postgresql.org/
- La doc : http://www.postgresql.org/docs/
- Actualité : http://planet.postgresql.org/

Le site officiel de la communauté se trouve sur http://www.postgresql.org/. Ce site contient des informations sur PostgreSQL, la documentation des versions maintenues, les archives des listes de discussion, etc.

Le site « Planet PostgreSQL » est un agrégateur réunissant les blogs des *core hackers*, des contributeurs, des traducteurs et des utilisateurs de PostgreSQL.

Serveurs francophones

- Site officiel : http://www.postgresql.fr/
- La doc : http://docs.postgresql.fr/
- Forum : http://forums.postgresql.fr/
- Actualité : http://planete.postgresql.fr/
- Wiki : http://wiki.postgresql.fr/

Le site postgresql.fr est le site de l'association des utilisateurs francophones du logiciel. La communauté francophone se charge de la traduction de toutes les documentations.

La console psql

- Un outil simple pour
  - les opérations courantes,
  - les tâches de maintenance,
  - les tests.

```
postgres$ psql
  base=#
```

La console `psql` permet d'effectuer l'ensemble des tâches courantes d'un utilisateur de bases de données. Si ces tâches peuvent souvent être effectuées à l'aide d'un outil graphique, la console présente l'avantage de pouvoir être utilisée en l'absence d'environnement graphique ou de scripter les opérations à effectuer sur la base de données.

Nous verrons également qu'il est possible d'administrer la base de données depuis cette console.

Enfin, elle permet de tester l'activité du serveur, l'existence d'une base, la présence d'un langage de programmation…

Obtenir de l'aide et quitter

- Obtenir de l'aide sur les commandes internes `psql`
  - `\? [motif]`
- Obtenir de l'aide sur les ordres SQL
  - `\h [motif]`
- Quitter
  - `\q` ou `ctrl-D`

`\h [NOM]` affiche l'aide en ligne des commandes SQL. Sans argument, la liste des commandes disponibles est affichée.

Exemple :

```
postgres=# \h savepoint
Command:     SAVEPOINT
Description: define a new savepoint within the current transaction
Syntax:
SAVEPOINT savepoint_name
```

`\q` ou `Ctrl-D` permettent de quitter la console.

Gestion de la connexion

- Obtenir des informations sur la connexion courante:
  - `\conninfo`
- Se connecter à une autre base
  - `\connect [DBNAME|- USER|- HOST|- PORT|-]`
  - `\c [DBNAME|- USER|- HOST|- PORT|-]`
- Modifier le mot de passe d'un utilisateur
  - `\password [USERNAME]`

`\c` permet de changer d'utilisateur et/ou de base de données sans quitter le client.

Exemple :

```
postgres@serveur_pg:~$ psql -q postgres
postgres=# \c formation stagiaire1
You are now connected to database "formation" as user "stagiaire1".
formation=> \c - stagiaire2
You are now connected to database "formation" as user "stagiaire2".
formation=> \c prod admin
You are now connected to database "prod" as user "admin".
```

Le gros intérêt de `\password` est d'envoyer le mot de passe chiffré au serveur. Ainsi, même si les traces contiennent toutes les requêtes SQL exécutées, il est impossible de retrouver les mots de passe via le fichier de traces. Ce n'est pas le cas avec un `CREATE USER` ou un `ALTER USER` (à moins de chiffrer soi-même le mot de passe).

Gestion de l'environnement système

- Chronométrer les requêtes
  - `\timing`
- Exécuter une commande OS
  - `\! [COMMAND]`

- Changer de répertoire courant
  - `\cd [DIR]`

`\timing` active le chronométrage des commandes. Il accepte un argument indiquant la nouvelle valeur (soit on, soit off). Sans argument, la valeur actuelle est basculée.

`\! [COMMANDE]` ouvre un shell interactif en l'absence d'argument ou exécute la commande indiquée.

`\cd` permet de changer de répertoire courant. Cela peut s'avérer utile lors de lectures ou écritures sur disque.

Catalogue système: objets utilisateurs

- Lister les bases de données
  - `\l`
- Lister les schémas
  - `\dn`
- Lister les fonctions
  - `\df[+] [motif]`

Ces commandes permettent d'obtenir des informations sur les objets utilisateurs : tables, index, vues, séquences, fonctions, agrégats, etc. stockés dans la base de données.

Pour les commandes qui acceptent un motif, celui-ci permet de restreindre les résultats retournés à ceux dont le nom d'opérateur correspond au motif précisé.

`\l[+]` dresse la liste des bases de données sur le serveur. Avec +, les commentaires et les tailles des bases sont également affichés.

`\d[+] [motif]` permet d'afficher la liste des tables de la base lorsqu'aucun motif n'est indiqué. Dans le cas contraire, la table précisée est décrite. Le + permet d'afficher également les commentaires associés aux tables ou aux lignes de la table, ainsi que la taille de chaque table.

`\db [motif]` dresse la liste des tablespaces actifs sur le serveur.

`\d{t|i|s|v}[S] [motif]` permet d'afficher respectivement :

- la liste des tables de la base active ;

- la liste des index ;
- la liste des séquences ;
- la liste des vues ;
- la liste des tables systèmes.

`\da` dresse la liste des fonctions d'agrégats.

`\df` dresse la liste des fonctions.

Catalogue système: rôles et accès

- Lister les rôles
  - `\du[+]`
- Lister les droits d'accès
  - `\dp`
- Lister les droits d'accès par défaut
  - `\ddp`
- Lister les configurations par rôle et par base
  - `\drds`

L'ensemble des informations concernant les objets, les utilisateurs, les procédures… sont accessibles par des commandes internes débutant par `\d`.

Pour connaître les rôles stockés en base, cette commande est `\du` (u pour user) ou `\dg` (g pour group). Dans les versions antérieures à la 8.0, les groupes et les utilisateurs étaient deux notions distinctes. Elles sont aujourd 'hui regroupées dans une notion plus générale, les rôles.

Les droits sont accessibles par les commandes `\dp` (p pour permissions) ou `\z` .

La commande `\ddp` permet de connaître les droits accordés par défaut à un utilisateur sur les nouveaux objets avec l'ordre `ALTER DEFAULT PRIVILEGES`.

```
cave=# \ddp
            Default access privileges
  Owner  | Schema | Type  |    Access privileges
---------+--------+-------+-----------------------
```

```
 caviste |        | table | caviste=arwdDxt/caviste+
         |        |       | u1=r/caviste
(1 row)
```

Enfin, la commande `\drds` permet d'obtenir la liste des paramètres appliqués spécifiquement à un utilisateur ou une base de données.

```
cave=# \drds
                List of settings
  Role   | Database |          Settings
---------+----------+---------------------------
 caviste |          | maintenance_work_mem=256MB
         | cave     | work_mem=32MB
(2 rows)
```

Catalogue système: tablespaces et extensions

- Lister les tablespaces
  - `\db`
- Lister les extensions
  - `\dx`

Visualiser le code des objets

- Code d'une vue
  - `\sv`
- Code d'une procédure stockée
  - `\sf`

Ceci permet de visualiser le code de certains objets sans avoir besoin de l'éditer.

Exécuter des requêtes

- Exécuter une requête
  - terminer une requête par `;`
  - ou par `\g`
  - ou encore par `\gx`
- Rappel des requêtes:
  - flèche vers le haut
  - `ctrl-R` suivi d'un extrait de texte représentatif

Sauf si `psql` est exécuté avec l'option `-S` (mode *single-line*), toutes les requêtes SQL doivent se terminer par `;` ou, pour marquer la parenté de PostgreSQL avec Ingres, `\g`.

En version 10, il est aussi possible d'utiliser `\gx` pour avoir l'action conjuguée de `\g` (ré-exécution de la requête) et de `\x` (affichage étendu).

La console `psql`, lorsqu'elle est compilée avec la bibliothèque libreadline ou la bibliothèque libedit (cas des distributions Linux courantes), dispose des mêmes possibilités de rappel de commande que le shell bash.

Afficher le résultat d'une requête

- Affichage par paginateur si l'écran ne suffit pas
- `\x` pour afficher un champ par ligne

En mode interactif, `psql` cherche d'abord à afficher directement le résultat :

```
postgres=# SELECT relname,reltype, relchecks, oid,oid FROM pg_class LIMIT 5;
              relname              | reltype | relchecks | oid  | oid
-----------------------------------+---------+-----------+------+----
--
 pg_statistic                      |   11258 |         0 | 2619 | 2619
 pg_type                           |      71 |         0 | 1247 | 1247
 pg_toast_2604                     |   11515 |         0 | 2830 | 2830
 pg_toast_2604_index               |       0 |         0 | 2831 | 2831
 pg_toast_2606                     |   11516 |         0 | 2832 | 2832
```

S'il y a trop de colonnes, on peut préférer n'avoir qu'un champ par ligne grâce au commutateur `\x` :

```
postgres=# \x on
postgres=# SELECT relname,reltype, relchecks, oid FROM pg_class LIMIT 3;
-[ RECORD 1 ]------------------
relname   | pg_statistic
reltype   | 11258
relchecks | 0
oid       | 2619
-[ RECORD 2 ]------------------
relname   | pg_type
reltype   | 71
relchecks | 0
oid       | 1247
-[ RECORD 3 ]------------------
relname   | pg_toast_2604
reltype   | 11515
relchecks | 0
oid       | 2830
```

`\x on` et `\x off` sont alternativement appelés si l'on tape plusieurs fois `\x`. `\x auto` délègue à `psql` la décision du meilleur affichage, en général à bon escient.

S'il n'y a pas la place à l'écran, `psql` appelle le paginateur par défaut du système. Il s'agit souvent de `more`, parfois de `less`. Ce dernier est bien plus puissant, permet de parcourir le résultat en avant et en arrière, avec la souris, de chercher une chaîne de caractères, de tronquer les lignes trop longues (avec l'option −S) pour naviguer latéralement.

Le paramétrage du paginateur s'effectue par des variables d'environnement :

```
export PAGER='less -S'
psql
```

ou :

```
PAGER='less -S'
```

ou dans `psql` directement, ou `.psqlrc` :

```
\setenv PAGER 'less -S'
```

Entrées/sorties

- Charger et exécuter un script SQL
  - `\i FICHIER`
- Rediriger la sortie dans un fichier
  - `\o FICHIER`
- Écrire un texte sur la sortie standard
  - `\echo texte...`
- Écrire un texte dans le fichier
  - `\qecho texte...`

`\i FICHIER` lance l'exécution des commandes placées dans le fichier passé en argument. `\ir` fait la même chose sauf que le chemin est relatif au chemin courant.

`\o [FICHIER | |COMMANDE]` envoie les résultats de la requête vers le fichier indiqué ou vers la commande UNIX au travers du tube.

Exemple :

```
postgres=# \o |a2ps
postgres=# SELECT ... ;
```

`\echo [TEXTE]` affiche le texte passé en argument sur la sortie standard.

`\qecho [TEXTE]` offre le même fonctionnement que `\echo [TEXTE]`, à ceci près que la sortie est dirigée sur la sortie des requêtes (fixée par `\o [ FICHIER]`) et non sur la sortie standard.

Exécuter un script SQL avec psql

- Exécuter un seul ordre SQL
  - `-c "ordre SQL"`
- Spécifier un script SQL en ligne de commande
  - `-f nom_fichier.sql`

- Possible de les spécifier plusieurs fois
  - exécutés dans l'ordre d'apparition
  - à partir de la version 9.6
- Charger et exécuter un script SQL depuis psql
  - `\i nom_fichier.sql`

L'option `-c` permet de spécifier la requête SQL en ligne de commande. Lorsque ce paramètre est utilisé, il implique automatiquement l'option `--no-psqlrc` jusqu'à la version 9.6.

Il est cependant souvent préférable de les enregistrer dans des fichiers si on veut les exécuter plusieurs fois sans se tromper. L'option `-f` est très utile dans ce cas.

Gestion des transactions

- `psql` est en mode auto-commit par défaut
  - variable `AUTOCOMMIT`
- Ouvrir une transaction explicitement
  - `BEGIN;`
- Terminer une transaction
  - `COMMIT;` ou `ROLLBACK;`
- Ouvrir une transaction implicitement
  - option `-1` ou `--single-transaction`

Par défaut, `psql` est en mode auto-commit, c'est-à-dire que tous les ordres SQL sont automatiquement validés après leur exécution.

Pour exécuter une suite d'ordres SQL dans une seule et même transaction, il faut soit ouvrir explicitement une transaction avec `BEGIN;` et la valider avec `COMMIT;` ou l'annuler avec `ROLLBACK;`.

Une autre possibilité est d'utiliser `psql -1` ou `psql --single-transation` afin d'ouvrir une transaction avant le début de l'exécution du script et de faire un `COMMIT` implicite à la fin de l'exécution du script, ou un `ROLLBACK` explicite le cas échéant. La présence d'ordres `BEGIN`, `COMMIT` ou `ROLLBACK` modifiera ce comportement en conséquence.

Gestion des erreurs

- Ignorer les erreurs dans une transaction
  - `ON_ERROR_ROLLBACK`
- Gérer des erreurs SQL en shell
  - `ON_ERROR_STOP`

La variable interne `ON_ERROR_ROLLBACK` n'a de sens que si elle est utilisée dans une transaction. Elle peut prendre trois valeurs :

- `off` (défaut) ;
- `on` ;
- `interactive`.

Lorsque `ON_ERROR_ROLLBACK` est à `on`, psql crée un `SAVEPOINT` systématiquement avant d'exécuter une requête SQL. Ainsi, si la requête SQL échoue, psql effectue un `ROLLBACK TO SAVEPOINT` pour annuler cette requête. Sinon il relâche le `SAVEPOINT`.

Lorsque `ON_ERROR_ROLLBACK` est à `interactive`, le comportement de psql est le même seulement si il est utilisé en interactif. Si psql exécute un script, ce comportement est désactivé. Cette valeur permet de se protéger d'éventuelles fautes de frappe.

Utiliser cette option n'est donc pas neutre, non seulement en terme de performances, mais également en terme d'intégrité des données. Il ne faut donc pas utiliser cette option à la légère.

Enfin, la variable interne `ON_ERROR_STOP` a deux objectifs : arrêter l'exécution d'un script lorsque psql rencontre une erreur et retourner un code retour shell différent de 0. Si cette variable reste à `off`, psql retournera toujours la valeur 0 même s'il a rencontré une erreur dans l'exécution d'une requête. Une fois activée, psql retournera un code d'erreur 3 pour signifier qu'il a rencontré une erreur dans l'exécution du script.

L'exécution d'un script qui comporte une erreur retourne le code 0, signifiant que psql a pu se connecter à la base de données et exécuté le script :

```
$ psql -f script_erreur.sql postgres
psql:script_erreur.sql:1: ERROR:  relation "vin" does not exist
LINE 1: SELECT * FROM vin;
                      ^
$ echo $?
0
```

Lorsque la variable ON_ERROR_STOP est activée, psql retourne un code erreur 3, signifiant qu'il a rencontré une erreur

```
$ psql -v ON_ERROR_STOP=on -f script_erreur.sql postgres
psql:script_erreur.sql:1: ERROR:  relation "vin" does not exist
LINE 1: SELECT * FROM vin;
                      ^
$ echo $?
3
```

psql retourne les codes d'erreurs suivant au shell :

- 0 au shell s'il se termine normalement ;
- 1 s'il y a eu une erreur fatale de son fait (pas assez de mémoire, fichier introuvable) ;
- 2 si la connexion au serveur s'est interrompue ou arrêtée ;
- 3 si une erreur est survenue dans un script et si la variable ON_ERROR_STOP a été initialisée.

Formatage des résultats

- Afficher des résultats non alignés
  - `-A | --no-align`
- Afficher uniquement les lignes
  - `-t | --tuples-only`
- Utiliser le format étendu
  - `-x | --expanded`
- Utiliser une sortie au format HTML
  - `-H | --html`
- Positionner un attribut de tableau HTML
  - `-T TEXT | --table-attr TEXT`

Séparateurs de champs

- Modifier le séparateur de colonnes
  - `-F CHAINE | --field-separator CHAINE`
- Forcer un octet 0x00 comme séparateur de colonnes
  - `-z | --field-separator-zero`
- Modifier le séparateur de lignes
  - `-R CHAINE | --record-separator CHAINE`
- Forcer un octet 0x00 comme séparateur de lignes
  - `-0 | --record-separator-zero`

`-F CHAINE` permet de modifier le séparateur de champ. Par défaut, il s'agit du caractère '|'. Cette option ne fonctionne qu'utilisée conjointement au modificateur de non-alignement des champs.

Exemple :

```
postgres@serveur_pg:~$ psql -F';' -A
postgres=# \l
List of databases
Name;Owner;Encoding;Collate;Ctype;Access privileges
b1;postgres;UTF8;C;C;
cave;caviste;UTF8;C;C;
module_C1;postgres;UTF8;C;C;
postgres;postgres;UTF8;C;C;
prod;postgres;UTF8;C;C;
template0;postgres;UTF8;C;C;=c/postgres
postgres=CTc/postgres
template1;postgres;UTF8;C;C;=c/postgres
postgres=CTc/postgres
(7 rows)
```

`-R CHAINE` permet de modifier le séparateur d'enregistrements. il s'agit par défaut du retour chariot. Ce commutateur ne fonctionne qu'utilisé conjointement au mode non-aligné de sortie des enregistrements.

Exemple :

```
postgres@serveur_pg:~$ psql -R' #Mon_séparateur# ' -A
postgres=# \l
List of databases #Mon_séparateur# Name|Owner|Encoding|Collate|Ctype|
Access privileges #Mon_séparateur# b1|postgres|UTF8|C|C| #Mon_séparateur# cave|
caviste|UTF8|C|C| #Mon_séparateur# module_C1|postgres|UTF8|C|C| #Mon_séparateur#
postgres|postgres|UTF8|C|C| #Mon_séparateur# template0|postgres|UTF8|C|C|
=c/postgres postgres=CTc/postgres #Mon_séparateur# template1|postgres|UTF8|C|C|
=c/postgres postgres=CTc/postgres #Mon_séparateur# (7 rows)
```

Les options `-z` et `-0` modifient le caractère nul comme séparateur de colonne et de ligne.

Travaux Dirigés 1

- Rappels de réseau
- Rappels de système
- Préparation du TP

Question de cours

- Quelle est la licence de PostgreSQL ?
- Quelle est la dernière version majeure de PostgreSQL ?
- Quel mot clé permet de valider une transaction ?

Rappels de réseau

- Client-serveur
- TCP/IP
- Port
- Socket Unix

Client-serveur L'environnement client-serveur désigne un mode de communication à travers un réseau entre plusieurs programmes :

- l'un, qualifié de client, envoie des requêtes ;
- l'autre ou les autres, qualifiés de serveurs, attendent les requêtes des clients et y répondent.

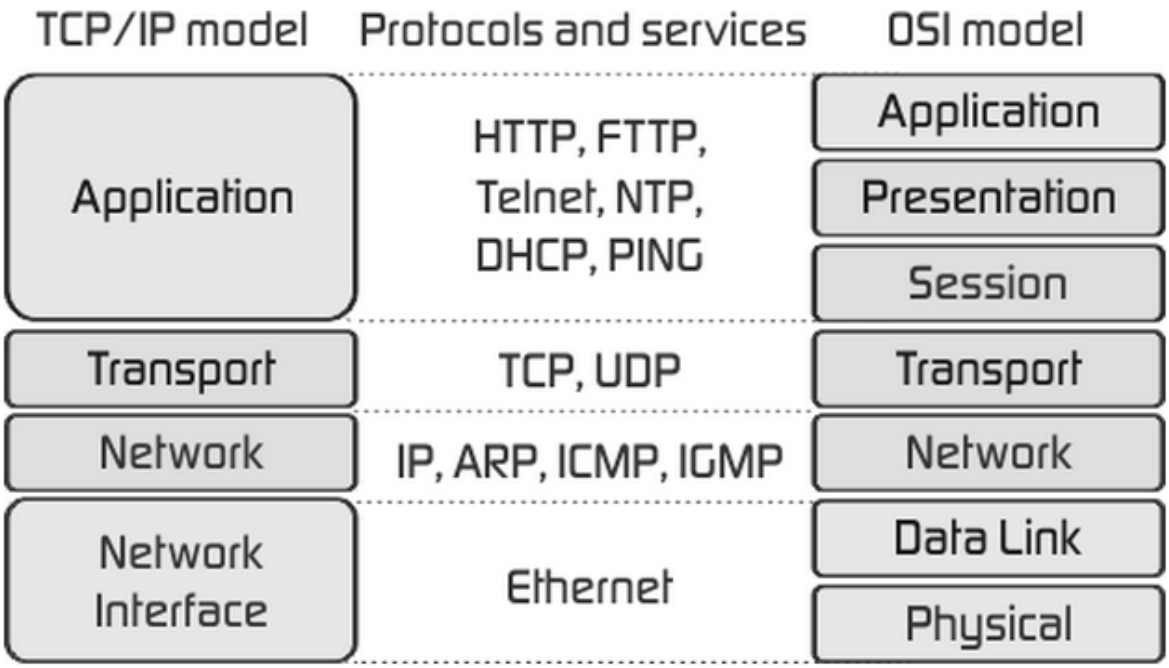
Caractéristiques d'un programme serveur :

- il attend une connexion entrante sur un ou plusieurs ports réseaux locaux ;
- à la connexion d'un client sur le port en écoute, il ouvre un socket local au système d'exploitation ;
- à la suite de la connexion, le processus serveur communique avec le client suivant le protocole prévu par la couche application du modèle OSI.

Caractéristiques d'un programme client :

- il établit la connexion au serveur à destination d'un ou plusieurs ports réseaux ;
- lorsque la connexion est acceptée par le serveur, il communique comme le prévoit la couche application du modèle OSI.

TCP/IP La suite TCP/IP est l'ensemble des protocoles utilisés pour le transfert des données sur Internet.



PostgreSQL utilise le protocole TCP pour échanger des données entre le client et le serveur.
TCP est un protocole de transport « fiable ». Il fournit un flux d'octets fiable assurant l'arrivée des données sans altérations et dans l'ordre, avec retransmission en cas de perte, et élimination des données dupliquées.

Port Nous avons vu qu'un programme serveur attend une connexion entrante sur un ou plusieurs ports réseaux locaux.

Pour simplifier, on peut considérer les ports comme des portes donnant accès au système d'exploitation. Pour fonctionner, un programme ouvre des portes pour entrer dans le système d'exploitation.
Grâce à cette abstraction, on peut exécuter plusieurs logiciels serveurs sur une même machine.

Un numéro de port est codé sur 16 bits, ce qui fait qu'il existe un maximum de 2^16 soit 65 536 ports distincts par machine. Ces ports sont classés en 3 catégories en fonction de leur numéro :

- les numéros de port de 0 à 1 023 correspondent aux ports “bien-connus” (well-known ports), utilisés pour les services réseaux les plus courants. Sur Linux, ils ne peuvent être ouverts par défaut que par l’utilisateur *root*.
- les numéros de ports de 1 024 à 49 151 correspondent aux ports enregistrés (registered ports), assignés par l’IANA.
- les numéros de ports de 49 152 à 65 535 correspondent aux ports dynamiques, utilisables pour tout type de requêtes TCP ou UDP autres que celle citées précédemment.

Le port par défaut de PostgreSQL est le 5432.

Socket Unix Les sockets du domaine UNIX utilisent le système de fichiers comme espace de noms. Cela permet à deux processus d’ouvrir le même socket pour communiquer. Toutefois, la communication se produit entièrement dans le noyau du système d’exploitation.

Il est possible d’établir une connexion à l’instance PostgreSQL en local en utilisant des socket Unix.

Rappels de système

- Processus
- systemd
- Bash
- psql
- OpenSSH
- Éditeur de fichier

Bases un système d’exploitation (souvent appelé OS — de l’anglais *Operating System*) est un ensemble de programmes qui dirige l’utilisation des ressources d’un ordinateur par des logiciels applicatifs. Il reçoit des demandes d’utilisation des ressources de l’ordinateur :

- ressources de stockage des mémoires : accès à la mémoire vive, aux disques durs,
- ressources de calcul du processeur central,
- ressources de communication vers des périphériques ou via le réseau.

Le système d’exploitation gère les demandes ainsi que les ressources nécessaires, évitant les interférences entre les logiciels.

Processus Un processus est un programme en cours d'exécution. Par exemple, chaque fois que l'on lance la commande `ls`, un processus est créé durant l'exécution de la commande.

Un processus est identifié par un numéro unique que l'on appelle le **PID** (Process IDentifiant).

Un processus dispose d'un processus père que l'on appelle le **PPID** (Parent PID).

On peut lister tous les processus actif d'un serveur grâce à la commande `ps`. Cette commande possède de nombreuses options. Se rapporter à sa page de manuel pour plus d'information.

On peut interagir avec un processus à travers des signaux. Ceux-ci offrent un mécanisme permettant d'envoyer un message à un processus en cours d'exécution. On se sert généralement des signaux pour terminer un processus, lui indiquer de relire sa configuration, etc.

systemd systemd est un système d'initialisation et un daemon qui a été spécifiquement conçu pour le noyau Linux.
Il est chargé de monter ou démonter les points de montage et de démarrer ou éteindre l'ensemble des services du serveur.
C'est une pièce maîtresse de l'architecture GNU/Linux. En effet, c'est le premier programme lancé par le noyau (il a donc le PID numéro 1) et il se charge de lancer tous les programmes suivants en ordre jusqu'à obtenir un système opérationnel pour l'utilisateur, selon le mode déterminé (single user, multi-user, graphique). C'est également à lui qu'incombe la tache de redémarrer ou arrêter votre ordinateur proprement.

L'outil de gestion des services dans systemd est `systemctl`. Il est généralement utilisé dans un terminal.

```
systemctl ACTION <Nom_du_service>.service
```

avec *ACTION* parmi :

- start : démarrer le service
- stop : arrêter le service
- restart : relancer le service
- reload : recharger le service
- status : connaitre l'état du service

Et : le nom du service visé.

Dans Debian, chaque instance sera contrôlée par un service avec la convention de nommage : `postgresql@<version>-<nom_instance>.service`

Bash Bash (acronyme de Bourne-Again shell) est un interpréteur en ligne de commande de type script. C'est le shell Unix du projet GNU.

Il se présente sous la forme d'une interface en ligne de commande accessible depuis la console ou un terminal. L'utilisateur lance des commandes sous forme d'une entrée texte exécutée ensuite par le shell.

Une bonne maîtrise de Bash est primordiale pour administrer et travailler avec PostgreSQL. La lecture de la page wikipedia[34], voir du man[35] est recommandée.

Bash fournit de nombreux outils permettant d'interagir avec le serveur. Le tutoriel suivant de Ubuntu[36] vous présente les commandes de base que vous devez impérativement connaître pour suivre ce cours.

La meilleure façon d'obtenir des informations sur un outil est d'utiliser son manuel grâce à la commande `man`.

psql La console `psql` permet d'effectuer l'ensemble des tâches courantes d'un utilisateur de bases de données. Si ces tâches peuvent souvent être effectuées à l'aide d'un outil graphique, la console présente l'avantage de pouvoir être utilisée en l'absence d'environnement graphique ou de scripter les opérations à effectuer sur la base de données.

OpenSSH OpenSSH est un ensemble d'outils informatiques libres permettant des communications sécurisées sur un réseau informatique en utilisant le protocole de communication sécurisé *SSH*.

La commande `ssh` est un programme permettant de se connecter à un système distant de façon sécurisé et d'y exécuter des commandes.
Nous l'utiliserons pour nous connecter à la machine virtuelle :

```
ssh <login>@<hostname>
```

La commande `scp` est un programme permettant le transfert sécurisé de fichiers entre deux ordinateurs.
Nous l'utiliserons pour nous connecter à la machine virtuelle.

Pour copier un fichier local sur un serveur distant :

```
scp <fichier> <login>@<hostname>:<chemin>
```

Pour copier un fichier d'un serveur distant en local :

[34] https://fr.wikipedia.org/wiki/Bourne-Again_shell
[35] https://linux.die.net/man/1/bash
[36] https://doc.ubuntu-fr.org/tutoriel/console_commandes_de_base

```
scp <login>@<hostname>:<fichier> <chemin>
```

Le nom d'hôte peut être une adresse IP ou bien un *FQDN*.

Éditeur de fichier La plupart des serveurs de production ne possèdent pas de session graphique. L'utilisation d'éditeur de fichier graphique est donc impossible. Il existe un certain nombre d'outils permettant d'éditer des fichiers en ligne de commande : *vi*, *nano*, *emacs*, …

Vous devrez impérativement utiliser un éditeur de ce type pour effectuer des modifications de configuration sur le serveur hébergeant votre instance PostgreSQL.

Préparation du TP

- Utilisation de VirtualBox
- Travail sur la base de données *cave*

La virtualisation consiste à exécuter sur une machine hôte dans un environnement isolé des systèmes d'exploitation ou des applications.

Lors des TP, nous utiliserons une machine virtuelle Debian sur VirtualBox.

VirtualBox est un hyperviseur de type 2. Il s'agit d'un logiciel qui tourne sur l'OS hôte. Ce logiciel permet de lancer un ou plusieurs OS invités. La machine virtualise ou/et émule le matériel pour les OS invités : ces derniers croient dialoguer directement avec ledit matériel.

Cette solution est très comparable à un émulateur, et parfois même confondue. Cependant l'unité centrale de calcul, c'est-à-dire le microprocesseur, la mémoire système (RAM) ainsi que la mémoire de stockage (via un fichier) sont directement accessibles aux machines virtuelles, alors que sur un émulateur l'unité centrale est simulée, les performances en sont donc considérablement réduites par rapport à la virtualisation.

Cette solution isole bien les OS invités, mais elle a un coût en performance. Ce coût peut être très élevé si le processeur doit être émulé, comme cela est le cas dans l'émulation. En échange cette solution permet de faire cohabiter plusieurs OS hétérogènes sur une même machine grâce à une isolation complète. Les échanges entre les machines se font via les canaux standards de communication entre systèmes d'exploitation (TCP/IP et autres protocoles réseau), un tampon d'échange permet d'émuler des cartes réseaux virtuelles sur une seule carte réseau réelle.

Les TP seront effectués sur une machine virtuelle hébergée sur VirtualBox.

Nous utiliserons le système d'exploitation GNU/Linux Debian en version Stretch.

Plusieurs version de PostgreSQL seront installées.

La base *cave* sera chargée dans l'une d'entre elles.

Il faudra dans un premier temps créer la machine virtuelle à partir d'un template.

Vous pourrez ensuite démarrer votre copie personnelle, vous y connecter en ssh et effectuer les actions demandées.

Bibliographie

La majeure partie des explications de ce travaux dirigés sont issus des pages suivantes :

- https://fr.wikipedia.org/wiki/Suite_des_protocoles_Internet
- https://fr.wikipedia.org/wiki/Transmission_Control_Protocol
- https://fr.wikipedia.org/wiki/Client%E2%80%93serveur
- https://fr.wikipedia.org/wiki/Port_%28logiciel%29
- https://fr.wikipedia.org/wiki/Berkeley_sockets#Socket_unix
- https://fr.wikipedia.org/wiki/Virtualisation
- https://fr.wikipedia.org/wiki/Syst%C3%A8me_d%27exploitation
- https://doc.ubuntu-fr.org/systemd
- https://fr.wikipedia.org/wiki/Secure_Shell

Solutions du TD 1

Question de cours

- Quelle est la licence de PostgreSQL ?
  - La licence Open Source PostgreSQL[37]. Elle permet d'utiliser, de copier, de modifier et de distribuer le logiciel et sa documentation pour n'importe quel usage gratuitement et sans accord écrit préalable.
- Quelle est la dernière version majeure de PostgreSQL ?
  - La dernière version majeure, sortie le 3 octobre 2019 est la 12.

[37] https://www.postgresql.org/about/licence/

- Quel mot clé permet de valider une transaction ?
  - le mot clé `COMMIT` permet de terminer une transaction en demandant la validation de l'ensemble des opérations au serveur.

Travaux Pratiques 1

- prise en main de PostgreSQL

Durant ces travaux pratiques, nous allons mettre en place notre serveur de base de données PostgreSQL via une machine virtuelle.

Effectuez les manipulations nécessaires pour réaliser les actions listées dans la section *Énoncés*.

Lorsque les commandes commencent par un dollar (`$`), il s'agit de commande à exécuter dans votre terminal bash avec votre utilisateur courant, soit sur la machine hôte (votre ordinateur), soit sur la machine virtuelle à travers une connexion ssh.
Si la commande débute par `#`, la commande shell est à lancer avec les droits super-utilisateur.

Les autres commandes sont des commandes à lancer dans `psql`.

> En cas de difficulté, se rapporter au cours, aux annexes de ce TP ainsi que de l'aide en ligne de PostgreSQL ou des pages de manuels (`man`).

Énoncés

Déployement de la machine virtuelle Nous allons commencer par créer une machine virtuelle.

Si vous êtes dans une des salles de TP à l'université, utiliser la première méthode : « TP sur place ». Si vous effectuez le TP à distance, rapportez vous à la seconde méthode.

TP sur place Listons les machines virtuelles disponibles :

```
$ virtualbox-createtp -l
```

Sélectionner la machine à créer (il y a *DB* dans le nom) :

```
$ virtualbox-createtp -c TP_bdd 2022-01-03_bullseye_DB_Admin
```

La commande suivante viendra modifier les paramètres de votre machine virtuelle afin qu'elle utilise 3 CPU et 2 Go de RAM :

```
$ vboxmanage modifyvm TP_bdd --cpus 3 --memory 2048
```

Pour lancer votre machine virtuelle, lancer la commande suivante dans votre terminal :

```
$ vboxmanage startvm TP_bdd --type headless
```

> A la fin de votre TP, n'oubliez pas de l'éteindre, elle restera sinon allumée et accessible à quiconque se connecte sur la machine que vous avez utilisée.

Vous pouvez éteindre votre machine virtuelle avec la commande :

```
$ vboxmanage controlvm TP_bdd acpipowerbutton
```

L'adresse IP est de votre machine virtuelle n'est pas toujours la même. Lancer la commande suivante pour la récupérer :

```
$ virtualbox-allocatedips
```

Vous devriez avoir un retour du type :

```
TP_bdd -> 192.168.56.102
```

Utilisez l'adresse IP récupérée par la commande ci-dessus pour vous connecter à votre machine virtuelle en ssh, utilisez la commande suivante :

```
$ ssh <login>@<IP_TP_bdd>
```

Les mots de passe sont identiques aux logins : *tp* et *root*.

Pour vous connecter au compte `postgres`, utilisez la commande à partir du compte *root* :

```
# sudo -iu postgres
```

Aide disponible sur la faq[38]

TP à distance Importer la machine virtuelle https://tribu-ml.fr/lstu/tp_bdd_ova_2022 dans Virtual Box. Renommer la machine virutelle en `TP_bdd`.

La commande suivante viendra modifier les paramètres de votre machine virtuelle afin qu'elle utilise 3 CPU et 2 Go de RAM :

```
$ vboxmanage modifyvm TP_bdd --cpus 3 --memory 2048
```

Pour lancer votre machine virtuelle, lancer la commande suivante dans votre terminal :

```
$ vboxmanage startvm TP_bdd --type headless
```

[38] https://faq.info.unicaen.fr/logiciels:virtualbox

La commande suivante permet de créer une redirection de port pour permettre une connexion par ssh à la machine virtuelle depuis votre poste :

```
vboxmanage controlvm TP_bdd natpf1 "ssh,tcp,,2222,,22"
```

Vous pouvez éteindre votre machine virtuelle avec la commande :

```
$ vboxmanage controlvm TP_bdd acpipowerbutton
```

Une fois la machine virtuelle allumée, vous pourrez vous y connecter en ssh en utilisant une redirection de port :

```
$ ssh -p 2222 tp@127.0.0.1
```

Les mots de passe sont identiques aux logins : *tp* et *root*.

Pour vous connecter au compte `postgres`, utilisez la commande :

```
# sudo -iu postgres
```

Découverte de l'instance Le but de ce TP est de découvrir les différents fichiers de l'instance PostgreSQL ainsi que les moyens de lancer l'instance.

Effectuez ces manipulation à partir de l'utilisateur *root*.

1. Afficher le statut de l'instance avec la commande :

```
systemctl status postgresql@13-main.service
```

   - Quel est le PID du processus `postgres` ?

2. Lister les fichiers de votre instances :

```
ls /var/lib/postgresql/13/main/
```

   - Comment nomme-t-on ce répertoire ?
   - Y a-t-il un moyen de connaitre la version de l'instance ?

3. Lister les fichiers de configuration de votre instance :

```
ls /etc/postgresql/13/main/
```

   - Quel fichier contrôle les droits d'accès à l'instance ?

4. Quel est le port d'écoute de votre instance ?

5. Lister les processus de votre instances :

```
sudo -iu postgres ps -o pid,cmd fx
```

---

psql Le but de ce TP est d'acquérir certains automatismes dans l'utilisation de `psql`.

Effectuez ces manipulation à partir de l'utilisateur *postgres*.

1. Lancez la console en se connectant à la base *postgres*.
2. Affichez l'ensemble des tables systèmes.
3. Déconnectez-vous.
4. Créer la base de données `mondial`
5. Reconnectez-vous à la base *postgres*.
6. Sans vous déconnecter, connectez-vous à la base *mondial*.
7. Récupérer le script du schéma *mondial* :
   `wget -O schema_mondial.sql https://tribu-ml.fr/lstu/tp_mondial_schema`
8. Charger le script que vous venez de récupérer dans la base de données *mondial*.
9. Affichez la liste des tables de la base de données.
10. Affichez toutes les lignes de la table `city`. Que remarquez vous ?
11. Récupérer le script du schéma *mondial* :
    `wget -O data_mondial.sql https://tribu-ml.fr/lstu/tp_mondial_data`
12. Charger le script que vous venez de récupérer dans la base de données *mondial*.
13. Affichez de nouveaux toutes les lignes de la table `city`. Que remarquez vous ?
14. Afficher la quantité de mémoire partagée de votre instance (paramètre `shared_buffers`).
15. Modifier la quantité de mémoire partagée de votre instance (paramètre `shared_buffers` dans le fichier `postgresql.conf` pour atteindre un quart de votre RAM (aidez vous du résultat de la commande `free -m`).
16. Afficher de nouveaux la quantité de mémoire partagée de votre instance. Le nouveau paramétrage a-t-il été pris en compte ? Dans la négative, trouvez un moyen pour que l'instance utilise le nouveau paramétrage.
17. Créer un nouvel utilisateur `monde` avec le droit de connexion et un mot de passe.

18. Modifier le fichier `pg_hba.conf` pour permettre à l'utilisateur `monde` de se connecter à la base de données `mondial` avec la méthode d'authentification `md5` depuis votre poste de travail (votre machine hôte).
19. Créer le fichier `.pgpass` afin de vous connecter sans mot de passe avec l'utilisateur `monde` depuis votre poste de travail (votre machine hôte).
20. Effectuez une copie de la liste des montagnes dans le fichier `/tmp/liste_mountain.txt`.
21. Affichez l'aide de la commande permettant de créer une table à partir des informations contenue dans une autre.
22. Créez une table `largest_city` composée du nom du pays, du code du pays, du nom de la plus grande ville du pays, de la population de cette ville.
23. Affichez le répertoire courant. Changez le pour `/tmp`. Vérifiez que vous êtes bien dans ce répertoire.
24. Écrivez un fichier contenant deux requêtes. Exécutez ce fichier dans la base de données `mondial`.

Annexes

Administration via bash

- `createdb`: ajouter une nouvelle base de données
- `createuser`: ajouter un nouveau compte utilisateur
- `dropdb`: supprimer une base de données
- `dropuser`: supprimer un compte utilisateur

Chacune de ces commandes est un « alias », un raccourci qui permet d'exécuter des commandes SQL de manière plus simple.

Par exemple, la commande système `dropdb` est équivalente à la commande SQL `DROP DATABASE`. L'outil `dropdb` se connecte à la base de données nommée `postgres` et exécute l'ordre SQL. Enfin, elle se déconnecte.

La création d'une nouvelle base se fait en utilisant l'outil `createdb` et en lui indiquant le nom de la nouvele base. Par exemple, pour créer une base `b1`, il faudrait faire ceci :

```
postgres$ createdb b1
```

Options connexion à PostgreSQL Tous les logiciels se connectant aux instances PostgreSQL partagent les mêmes paramètres de connexions.

Les options de connexion permettent de préciser l'utilisateur et la base de données utilisés pour la connexion initiale.

Lorsque l'une des options n'est pas précisée, la bibliothèque cliente PostgreSQL vérifie qu'il n'existe pas une variable shell correspondante et prend sa valeur. S'il ne trouve pas de variable, il utilise une valeur par défaut.

`-h HOTE` ou `$PGHOST` permet de préciser l'alias ou l'adresse `IP` de la machine qui héberge le serveur. Sans indication, le client se connecte sur la socket Unix dans `/tmp`.

`-p PORT` ou `$PGPORT` permet de préciser le port sur lequel le serveur écoute les connexions entrantes. Sans indication, le port par défaut est le 5432.

`-U NOM` ou `$PGUSER` permet de préciser le nom de l'utilisateur utilisé pour la connexion. L'option `-U` n'est toutefois pas nécessaire, sous réserve que les arguments de la ligne de commande respectent l'ordre `NOM_BASE NOM_UTILISATEUR`. Sans indication, le nom d'utilisateur PostgreSQL est le nom de l'utilisateur utilisé au niveau système.

`-d base` ou `$PGDATABASE` permet de préciser le nom de la base de données utilisée pour la connexion. Le drapeau `-d` n'est pas nécessaire sous réserve que le nom de la base de données est le dernier argument de la ligne de commande. Sans indication le nom de la base de données de connexion correspondra au nom de l'utilisateur utilisé au niveau PostgreSQL.

Le tableau suivant résume ce qui est écrit plus haut :

option	variable	défaut
-h HOTE	$PGHOST	/tmp ou /var/run/postgresql
-p PORT	$PGPORT	5432
-U NOM	$PGUSER	nom de l'utilisateur OS
-d base	$PGDATABASE	nom de l'utilisateur PG

Authentification via `.pgpass` Le fichier `.pgpass`, situé dans le répertoire personnel de l'utilisateur ou celui référencé par `$PGPASSFILE`, est un fichier contenant les mots de passe à utiliser si la connexion requiert un mot de passe (et si aucun mot de passe n'a été spécifié).

Ce fichier devra être composé de lignes du format :

```
nom_hote:port:nom_base:nom_utilisateur:mot_de_passe
```

Chacun des quatre premiers champs pourraient être une valeur littérale ou `*` (qui correspond à tout). La première ligne réalisant une correspondance pour les paramètres de connexion sera utilisée (du coup, placez les entrées plus spécifiques en premier lorsque vous utilisez des jokers). Si une entrée a besoin de contenir `*` ou `\`, échappez ce caractère avec `\`. Un nom d'hôte `localhost` correspond à la fois aux connexions `host` (TCP) et aux connexions `local` (`socket` de domaine *Unix*) provenant de la machine locale.

Les droits sur `.pgpass` doivent interdire l'accès aux autres et au groupe ; réalisez ceci avec la commande `chmod 0600 ~/.pgpass`. Si les droits du fichier sont moins stricts, le fichier sera ignoré.

Droits de connexion Lors d'une connexion, indication :

- de l'hôte (socket Unix ou alias/adresse IP)
- du nom de la base de données
- du nom du rôle
- du mot de passe (parfois optionnel)

Lors d'une connexion, l'utilisateur fournit, explicitement ou non, plusieurs informations. PostgreSQL va choisir une méthode d'authentification en se basant sur les informations fournies et sur la configuration d'un fichier appelé `pg_hba.conf`.

`pg_hba.conf`

Se présente sous la forme d'un tableau :

- 4 colonnes d'informations
  - type
  - database
  - user
  - adresse
- 1 colonne indiquant la méthode à appliquer
- 1 colonne optionnelle d'options

Lorsque le serveur PostgreSQL récupère une demande de connexion, il connaît le type de connexion utilisé par le client (socket Unix, connexion TCP SSL, connexion TCP simple, etc). Il connaît aussi l'adresse IP du client (dans le cas d'une connexion via une socket TCP), le nom de la base et celui de l'utilisateur. Il va donc parcourir les lignes du tableau enregistré dans le fichier `pg_hba.conf`. Il les parcourt dans l'ordre. La première qui correspond aux informations fournies lui précise la méthode

d'authentification. Il ne lui reste plus qu'à appliquer cette méthode. Si elle fonctionne, la connexion est autorisée et se poursuit. Si elle ne fonctionne pas, quelle qu'en soit la raison, la connexion est refusée. Aucune autre ligne du fichier ne sera lue.

Il est donc essentiel de bien configurer ce fichier pour avoir une protection maximale.

Le tableau se présente ainsi :

```
# local      DATABASE  USER  METHOD  [OPTIONS]
# host       DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostssl    DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostnossl  DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
```

La colonne *type* peut contenir quatre valeurs différentes. La valeur `local` concerne les connexions via la socket Unix. La valeur `host`, concernent les connexions via la socket TCP.

La colonne *database* peut recueillir le nom d'une base, le nom de plusieurs bases en les séparant par des virgules, le nom d'un fichier contenant la liste des bases ou quelques valeurs en dur. La valeur all indique toutes les bases.

La colonne *user* peut recueillir le nom d'un rôle, le nom d'un groupe en le précédant d'un signe plus, le nom de plusieurs rôles en les séparant par des virgules, le nom d'un fichier contenant la liste des bases ou quelques valeurs en dur. La valeur all indique tous les rôles.

La colonne *adresse* est présente uniquement dans le cas d'une connexion de type `host`. Cette colonne permet d'indiquer une adresse IP ou un sous-réseau IP. Il est donc possible de filtrer les connexions par rapport aux adresses IP, ce qui est une excellente protection.

Voici deux exemples d'adresses IP au format adresse et masque de sous-réseau :

```
192.168.168.1 255.255.255.255
192.168.168.0 255.255.255.0
```

Et voici deux exemples d'adresses IP au format CIDR :

```
192.168.168.1/32
192.168.168.0/24
```

La colonne de `méthode` précise la méthode d'authentification à utiliser. Il existe 2 types de méthodes, *internes* ou *externes*.

Méthodes internes à privilégier : `md5`.

Les méthodes *externes* permettent d'utiliser des annuaires d'entreprise comme RADIUS, LDAP ou un ActiveDirectory. Certaines méthodes sont spécifiques à Unix (comme `ident` et `peer`), voire à Linux (comme `pam`).

Mise à jour des droits de connexion

Pour permettre la prise en compte des modifications sur le fichier `pg_hba.conf` il faut recharger la configuration de l'instance. Ceci peut se faire soit via *bash* avec la commande :

```
# systemctl reload postgresql
```

Ou via la console `psql` avec la commande SQL :

```
postgres=# SELECT pg_reload_conf();
 pg_reload_conf
----------------
 t
(1 ligne)
```

Solutions du TP 1

Découverte de l'instance

1. Processus ID : 1224

```
$ systemctl status postgresql@13-main.service
* postgresql@13-main.service - PostgreSQL Cluster 13-main
     Loaded: loaded (/lib/systemd/system/postgresql@.service;
      ↪  enabled-runtime; vendor preset: enabled)
     Active: active (running) since Thu 2022-01-06 19:26:06 CET; 26min ago
    Process: 17424 ExecStart=/usr/bin/pg_ctlcluster
     ↪  --skip-systemctl-redirect 13-main start (code=exited,
     ↪  status=0/SUCCESS)
   Main PID: 17429 (postgres)
      Tasks: 7 (limit: 1133)
     Memory: 643.6M
        CPU: 15.150s
     CGroup:
      ↪  /system.slice/system-postgresql.slice/postgresql@13-main.service
             ├─17429 /usr/lib/postgresql/13/bin/postgres \
                          -D /var/lib/postgresql/13/main \
                    -c config_file=/etc/postgresql/13/main/postgresql.conf
```

```
             ├─17431 postgres: 13/main: checkpointer
             ├─17432 postgres: 13/main: background writer
             ├─17433 postgres: 13/main: walwriter
             ├─17434 postgres: 13/main: autovacuum launcher
             ├─17435 postgres: 13/main: stats collector
             └─17436 postgres: 13/main: logical replication launcher

janv. 06 19:26:04 tp-bdd systemd[1]: Starting PostgreSQL Cluster 13-main...
janv. 06 19:26:06 tp-bdd systemd[1]: Started PostgreSQL Cluster 13-main.
```

2. Ce répertoire se nomme usuellement *PGDATA*. Le fichier `PG_VERSION` contient la version de l'instance.
3. Le fichier `pg_hba.conf` (*host-based authentication*) contrôle les droits d'accès à l'instance.
4. Le port d'écoute est celui par défaut : 5432. Il y a plusieurs moyen pour le récupérer :
   - la commande `netstat -anp | grep postgres` liste les ports ouverts. On recherche dans les résultats le processus ayant le PID trouvé plus haut.
   - la commande `psql -c 'SHOW port;'` va renvoyer l'information du port de l'instance par défaut.
5. L'utilisateur postgres n'est pas sudoer. Il faut soit retourner avec l'utilisateur root, soit lancer la commande `ps -o pid,cmd fx`

```
  17429 /usr/lib/postgresql/13/bin/postgres -D /var/lib/postgresql/13/main \
                   -c config_file=/etc/postgresql/13/main/postgresql.conf
  17431  \_ postgres: 13/main: checkpointer
  17432  \_ postgres: 13/main: background writer
  17433  \_ postgres: 13/main: walwriter
  17434  \_ postgres: 13/main: autovacuum launcher
  17435  \_ postgres: 13/main: stats collector
  17436  \_ postgres: 13/main: logical replication launcher
------
```

```
#### psql

1. `psql -d postgres` (avec l'utilisateur _postgres_ on obtient le même
   résultat avec la commande `psql`)

2. `\dS`

3. `\q` ou `<CRTL>+D`
```

4. On peut utiliser la commande _bash_ qui est l'équivalent de l'ordre SQL `CREATE DATABASE`

```bash
$ createdb mondial
```

5. psql
6. \c mondial
7. \! wget -O schema_mondial.sql https://tribu-ml.fr/lstu/tp_mondial_schema
8. \i <chemin vers le fichier du schéma mondial>
9. \d
10. La table est vide (la commande est SELECT * FROM city).
11. \! wget -O data_mondial.sql https://tribu-ml.fr/lstu/tp_mondial_data
12. \i <chemin vers le fichier des données mondial>
13. La table n'est plus vide.
14. SHOW shared_buffers;
15. Éditer le fichier */etc/postgresql/13/main/postgresql.conf*. Fixer le paramètre *shared_buffers* à 512MB.
16. Il est noté dans le fichier de configuration que le changement nécessite un redémarrage de l'instance (*change requires restart*). L'instance doit être redémarrée par l'utilisateur root avec la commande :
    systemctl restart postgresql@13-main.service
17. méthode via le terminal :

```
postgres$ createuser -P monde
```

méthode via psql :

```
CREATE ROLE monde LOGIN PASSWORD 'secret';
```

18. Ajouter la ligne suivante à votre fichier pg_hba.conf :

```
host    mondial             monde               <IP workstation>/32
md5
```

Recharger la configuration de votre instance via bash :

```
# systemctl reload postgresql
```

Ou via la console `psql` avec la commande SQL :

```
postgres=# SELECT pg_reload_conf();
 pg_reload_conf
----------------
 t
(1 ligne)
```

Cela ne suffit cepandant pas car votre serveur PostgreSQL n'écoute que sur *localhost*. Il vous faut mettre à jour le paramètre `listen_addresses` de votre fichier *postgresql.conf*. Le plus simple est de le fixer à `'*'`. Vous devez alors redémarrer l'isntance (en tant que *root*) :

```
systemctl restart postgresql@13-main.service
```

Puis vous pouvez vous connecter depuis votre poste de travail :

```
$ psql -h <IP VM> -U monde mondial
```

19. Insérer les lignes suivantes dans le fichier `~/.pgpass` sur votre machine hôte :

```
#hostname:port:database:username:password
*:5432:mondial:monde:secret
```

Changer les droits pour ce fichier :

```
$ chmod 600 ~/.pgpass
```

Se connecter à l'instance depuis le poste de travail :

```
$ psql -h <IP VM> -U monde mondial
```

20. 2 possibilités :

```
mondial=> \\o /tmp/liste_mountain.txt'
mondial=> SELECT  * FROM mountain;
```

Ou :

```
mondial=> \copy mountain to '/tmp/liste_mountain.txt'
```

21. `\h CREATE TABLE AS`

22.

```
CREATE TABLE largest_city AS
  SELECT c.name country,c.code,c1.name,c1.population
    FROM country c, city c1,
      (SELECT country,MAX(population) AS truc
```

```
        FROM city GROUP BY country) c2
    WHERE c1.population>= c2.truc
      AND c1.country = c2.country
        AND c1.country=c.code;
```

23.

```
\! pwd
\cd /tmp
\! pwd
```

24.

```
postgres$ vi requete.sql
postgres$ psql -f requete.sql mondial
```
