TP N° 1 — Initiation à PostgreSQL

(Sous Windows)

M. Laïdi FOUGHALI

l.foughali@univ-skikda.dz

(Le support ⇒ al-moualime.com)

Université de Skikda — Département d'Informatique

1re Année Master RSD/IA

Bases de Données Avancées (BDA)

23/10 - 12/11 2025

Version 1.0 (Initiale) — 25 octobre 2025 à 07:49:12

Plan

Qu'est-ce que PostgreSQL ?

PostgreSQL est un **Système de Gestion de Bases de Données Relationnelles-Objet (SGBDRO)**, issu du projet universitaire **POSTGRES (v4.2)** développé à l'Université de Californie à Berkeley.

Caractéristiques principales :

- Héritier libre du code original du projet académique de Berkeley.
- Conforme à une large partie du standard **SQL** (ISO/IEC 9075).
- Offre des fonctionnalités avancées :
  - Requêtes complexes, clés étrangères, déclencheurs (*triggers*).
  - Vues modifiables et intégrité transactionnelle complète.
  - Contrôle de concurrence multiversion (**MVCC**).

Extensibilité : PostgreSQL permet d'ajouter de nouveaux **types de données**, **fonctions**, **opérateurs**, **méthodes d'indexation**, ainsi que des **langages procéduraux** personnalisés (PL/pgSQL, PL/Python, etc.).

Licence PostgreSQL

Distribué sous une licence **libre et permissive**, PostgreSQL peut être utilisé, modifié et redistribué sans restriction, à des fins *privées*, *académiques* ou *commerciales*.

Bref historique de PostgreSQL

PostgreSQL est issu du projet **POSTGRES**, lancé en 1986 à l'Université de Californie à Berkeley sous la direction du professeur **Michael Stonebraker**.

Évolution historique :

- **POSTGRES (1986–1993)** — projet de recherche visionnaire introduisant des **règles actives** (ancêtres des *triggers*), des **transactions avancées** et un **stockage orienté objet**. Utilisé dans les *sciences*, la *médecine* et les *SIG*.
- **Postgres95 (1994)** — ajout du langage **SQL**, création du client **psql** et simplification du code source.
- **PostgreSQL (1996 → ...)** — adoption du nom actuel marquant la compatibilité totale avec SQL. Développement **ouvert et communautaire**, enrichi en continu.

Conclusion

Aujourd'hui, **PostgreSQL** s'impose comme le **SGBD libre de référence**, plébiscité aussi bien dans le milieu académique que professionnel.

Architecture client/serveur de PostgreSQL

PostgreSQL fonctionne selon un modèle **client/serveur** : le client envoie des requêtes SQL, le serveur les interprète, exécute les opérations demandées et renvoie les résultats.

Côté client : Les clients peuvent être des outils interactifs (`psql`, **pgAdmin**), des applications web ou des scripts. Ils communiquent avec le serveur via le protocole **TCP/IP** en utilisant un port par défaut (5432).

Côté serveur : Le processus principal `postgres` écoute les connexions entrantes. Lorsqu'un client se connecte, il crée un **processus fils dédié** qui gère exclusivement cette session. Ce processus reçoit la requête, la transmet au **parser**, qui la traduit en arbre syntaxique, puis au **planificateur/optimiseur**, qui détermine la stratégie d'exécution la plus efficace. Enfin, le **moteur d'exécution** applique ce plan et renvoie les résultats au client.

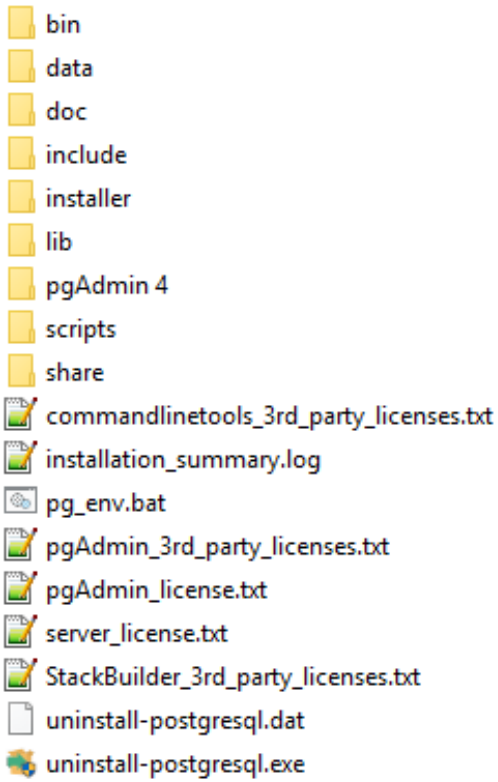
Avantage : Cette architecture permet un fonctionnement **concurrent**, stable et sécurisé, où chaque session est isolée, mais toutes partagent les mêmes fichiers de données gérés par le serveur principal.

Procédure d'installation (PostgreSQL version 16)

- **Téléchargement** : Site officiel → https://www.postgresql.org/download/windows (exécutable : `postgresql-16.x-windows-x64.exe`).
- **Exécution de l'installeur** : lancer en mode administrateur (`clic droit → Exécuter en tant qu'administrateur`).
- **Chemins d'installation** (à mémoriser) : `C:\Program Files\PostgreSQL\16\` (utile pour retrouver les fichiers binaires et de configuration).
- **Composants à installer** :
  - PostgreSQL Server
  - pgAdmin 4
  - Command Line Tools (`psql`)
  - Stack Builder (optionnel)
- **Configuration initiale** :
  - Mot de passe superutilisateur : `postgres`.
  - Port par défaut : 5432.
  - Service Windows créé : `postgresql-x64-16`.

Contenu de l'installation

Le contenu de l'installation se présente ainsi :



- **bin/** : exécutables principaux.
- **data/** : répertoire des données (cluster PostgreSQL).
- **doc/** : documentation locale.
- **include/** : fichiers d'en-tête pour le développement.
- **lib/** : bibliothèques dynamiques (DLL).
- **pgAdmin 4/** : interface graphique d'administration.
- **scripts/** : scripts utilitaires.
- **share/** : fichiers partagés (init, dictionnaires, modèles).
- **StackBuilder/** : installateur d'extensions complémentaires.

Fichiers associés : `installation_summary.log`, `pg_env.bat`, `server_license.txt`, `uninstall-postgresql.exe`, etc.

Dossier `bin` — Exécutables principaux

Le répertoire **bin/** regroupe les programmes essentiels utilisés pour **démarrer, administrer et interagir** avec le serveur PostgreSQL.

- **postgres.exe** — lance le serveur principal.
- **psql.exe** — client en ligne de commande.
- **pg_ctl.exe** — démarre, arrête ou redémarre le serveur.
- **initdb.exe** — initialise un cluster de bases.
- **pg_dump(.exe)** / **pg_dumpall(.exe)** — export et sauvegarde.
- **pg_restore.exe** — restauration depuis une sauvegarde.
- **createdb(.exe)** / **dropdb(.exe)** — création et suppression de bases.
- **createuser(.exe)** / **dropuser(.exe)** — gestion des rôles utilisateurs.
- **vacuumdb.exe** — nettoyage et optimisation des tables.
- **pgAdmin4.exe** — interface graphique d'administration.

Ces exécutables constituent la base de toute administration PostgreSQL sous Windows.

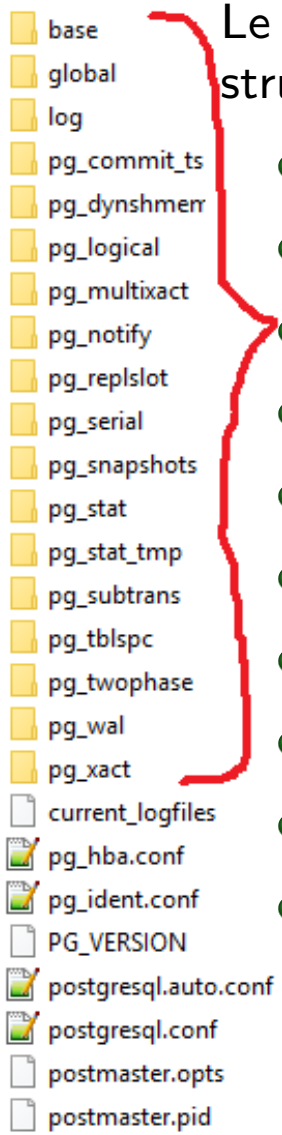
SQL Shell — `psql`

Le programme **psql.exe** est le client en ligne de commande officiel.

- **Fonctions** :
  - Connexion au serveur PostgreSQL.
  - Exécution de commandes SQL (`SELECT`, `INSERT`, `UPDATE`, etc.).
  - Commandes internes (`\d`, `\l`, `\c`, ...).
  - Exécution de scripts SQL (`\i fichier.sql`).
- **Avantages** : complet, rapide, puissant pour l'administration et les tests.
- **Accès** : depuis le menu Démarrer (SQL Shell) ou via terminal : `psql -U postgres -d nom_base`.

⇒ Outil principal pour apprendre et pratiquer le SQL sous PostgreSQL.

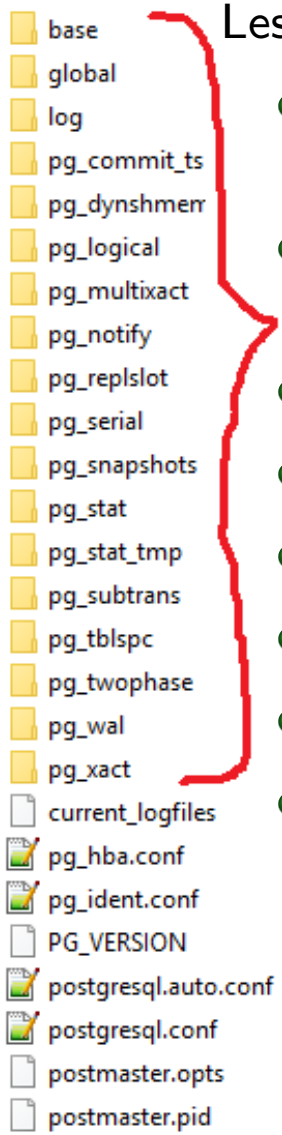
Cluster — data/



Le répertoire **data/** contient l'**ensemble du cluster PostgreSQL**, structuré en sous-dossiers essentiels :

- **base/** : stocke les données binaires des bases utilisateur.
- **global/** : contient les rôles et catalogues globaux du cluster.
- **pg_wal/** : journaux de transactions (Write Ahead Log).
- **pg_xact/** : états des transactions (commit/rollback).
- **pg_multixact/** : gestion des verrous partagés.
- **pg_stat/**, **pg_stat_tmp/** : statistiques du serveur.
- **pg_logical/** : support de la réplication logique.
- **pg_subtrans/** : suivi des sous-transactions imbriquées.
- **pg_twophase/** : transactions en deux phases (2PC).
- **pg_tblspc/** : liens vers des tablespaces externes.

Cluster - fichiers de configuration



Les fichiers essentiels du répertoire **data/** sont :

- **postgresql.conf** : configuration principale (port, mémoire, logs, etc.).
- **pg_hba.conf** : règles de contrôle des accès et méthodes d'authentification.
- **pg_ident.conf** : mappage utilisateurs système et PostgreSQL.
- **postgresql.auto.conf** : paramètres modifiés via `ALTER SYSTEM`.
- **PG_VERSION** : version de PostgreSQL utilisée.
- **postmaster.pid** : PID du processus serveur en cours d'exécution.
- **postmaster.opts** : options de lancement utilisées au démarrage.
- **current_logfiles** : emplacement des journaux actifs.

Objets de l'installation

Lors de l'installation, PostgreSQL crée automatiquement plusieurs éléments essentiels :

- **La base postgres** : base par défaut pour se connecter immédiatement.
- **Le rôle postgres** : rôle par défaut ayant tous les privilèges.
- **Les bases modèles (templates)** : servent de modèle pour créer de nouvelles bases.

Schéma sous PostgreSQL

Un **schéma** est un espace logique de rangement à l'intérieur d'une base de données. Il contient des objets comme des tables, vues, fonctions, séquences, etc. Par défaut, chaque base contient un schéma **public**.

Pour commencer avec PostgreSQL :

- Se connecter avec le rôle `postgres`.
- Travailler dans la base **postgres**.
- Créer de nouvelles bases à partir de **template0** ou **template1**.
- Organiser les objets dans des **schémas**.

La base postgres

Lors de l'installation, **PostgreSQL** crée automatiquement une base nommée **postgres**. Il s'agit d'une base spéciale servant d'**environnement de travail initial**.

Utilisations principales :

- Base de connexion par défaut (si aucune autre n'est spécifiée).
- Base de test et d'expérimentation pour exécuter des commandes simples.
- Base d'administration permettant la gestion des rôles et des bases dès la première connexion.

Bonnes pratiques :

- Ne jamais la supprimer : elle constitue une **base de secours** garantissant toujours une connexion au serveur.
- L'utiliser pour des essais rapides, mais créer des bases dédiées pour les projets applicatifs.

À retenir

La base `postgres` est une **base utilitaire et d'administration**, indispensable à conserver même lorsque d'autres bases sont créées.

Le rôle postgres

Lors de l'installation, **PostgreSQL** crée automatiquement un rôle spécial nommé **postgres**. Ce rôle correspond au **superutilisateur** du système.

Caractéristiques principales :

- Dispose de tous les privilèges : création de bases, gestion des rôles, configuration du serveur.
- Utilisé pour la configuration initiale et les premières connexions.
- Peut accéder au serveur via `psql`, **pgAdmin** ou tout autre client SQL compatible.

Bonnes pratiques :

- Éviter d'utiliser directement `postgres` pour les bases applicatives.
- Créer des rôles dédiés à chaque projet avec des droits limités et adaptés.
- Réserver `postgres` exclusivement aux tâches d'administration et de maintenance.

À retenir

Le rôle `postgres` est le **superutilisateur par défaut**. Il doit être utilisé uniquement pour l'administration et jamais pour le développement courant.

Les bases modèles template0 et template1

Lors de l'installation, **PostgreSQL** crée deux bases modèles servant de référence à toute nouvelle base. Chaque base créée est une **copie** de l'un de ces modèles.

template0 :

- Base minimale, intacte et jamais modifiée.
- Utilisée pour créer des bases totalement « propres », sans personnalisation.

template1 :

- Base modèle personnalisable.
- Les extensions, fonctions ou langues ajoutées ici sont recopiées dans les nouvelles bases.

Choix du modèle lors de la création :

```
CREATE DATABASE nom_base TEMPLATE template0;
```

```
CREATE DATABASE nom_base; -- utilise template1 par défaut
```

À retenir

Toute nouvelle base PostgreSQL est une copie de l'un de ces deux modèles.

Les rôles dans PostgreSQL

Dans **PostgreSQL**, il n'existe pas de distinction entre **utilisateur** et **groupe** : tout est représenté par un **rôle** (`ROLE`).

Principe fondamental

Un rôle peut :

- se connecter au serveur (comme un utilisateur) ;
- posséder des objets : bases, tables, vues, fonctions, etc. ;
- recevoir ou attribuer des privilèges à d'autres rôles ;
- regrouper d'autres rôles (fonction de groupe).

Remarque

Le rôle `postgres` est le **superutilisateur** créé automatiquement lors de l'installation. Il dispose de tous les privilèges et peut administrer l'ensemble du serveur.

Les rôles de connexion (LOGIN)

Un **rôle de connexion** est un rôle autorisé à se connecter au serveur **PostgreSQL**. Il correspond à un **utilisateur classique** du système.

```
-- Création d'un rôle de connexion (avec mot de passe)
CREATE ROLE vendeur LOGIN PASSWORD 'vendeur123';

-- Commande équivalente : CREATE USER est un alias de
--   CREATE ROLE ... LOGIN
CREATE USER vendeur PASSWORD 'vendeur123';
```

- Peut ouvrir une session via `psql`, `pgAdmin`, etc.
- Possède ses propres privilèges et objets.
- Sert généralement aux connexions applicatives.

Exemple

Le rôle `vendeur`, utilisé dans la base `pizzerias`, est un rôle de connexion.

Les rôles sans connexion (NOLOGIN)

Un **rôle sans connexion** n'a pas l'attribut `LOGIN`. Il sert à **regrouper plusieurs utilisateurs** afin de gérer leurs privilèges de manière collective.

```
-- Création d'un rôle "groupe"
CREATE ROLE employes;

-- Attribution du rôle "employes" à plusieurs utilisateurs
GRANT employes TO vendeur, serveur, gerant;
```

- Ne peut pas se connecter directement au serveur.
- Sert à simplifier la gestion des droits (principe d'héritage).
- Tous les membres héritent des privilèges accordés au groupe.

Exemple

Si le rôle `employes` possède un droit de lecture sur une table, alors `vendeur` et `serveur` l'héritent automatiquement.

[1/4] Schémas — Notion et utilité

Définition

Un **schéma** est un **espace de noms logique** à l'intérieur d'une base de données. Il regroupe les objets — **tables**, **vues**, **fonctions**, **séquences** — pour structurer et isoler les composants d'un même projet.

Un schéma permet de :

- **Organiser** la base par domaine fonctionnel (par exemple : *application* (`app`), *référentiel* (`ref`), *audit* (`audit`), *journal* (`log`), etc.).
- **Séparer** les droits et responsabilités selon les rôles ou utilisateurs.
- **Simplifier** la gestion et éviter les conflits de noms d'objets.

Idée clé

Un schéma agit comme un **dossier logique** à l'intérieur d'une même base : il apporte de la **clarté**, de la **structure** et de la **sécurité**, sans nécessiter la création de plusieurs bases distinctes.

[2/4] Le schéma par défaut — public

Lorsqu'une base est créée, PostgreSQL y ajoute automatiquement un schéma nommé **public**. Tous les utilisateurs ont, par défaut, le droit d'y créer des objets.

Exemple

```
-- Crée implicitement la table dans le schéma "public"
CREATE TABLE clients (id SERIAL PRIMARY KEY, nom TEXT);

-- Équivalent explicite
CREATE TABLE public.clients (id SERIAL PRIMARY KEY, nom TEXT);
```

Remarque

Si aucun schéma n'est précisé, PostgreSQL utilise **public**. Pour un projet structuré, il est conseillé de créer ses propres schémas applicatifs.

[3/4] Création et configuration

Un **schéma** a toujours un **propriétaire** (rôle) défini à sa création via la clause `AUTHORIZATION`, qui lui accorde tous les droits sur ses objets.
Le **search_path** indique l'ordre dans lequel PostgreSQL cherche les schémas lorsqu'un nom d'objet n'est pas qualifié (`schema.table`). Le premier schéma du chemin est utilisé par défaut.

Création d'un schéma

```
-- Création et définition du propriétaire
CREATE SCHEMA IF NOT EXISTS app AUTHORIZATION app_owner;
```

Chemin de recherche

```
-- Ordre de recherche : d'abord app, puis public
ALTER ROLE vendeur SET search_path TO app, public;

-- Afficher le chemin courant
SHOW search_path;
```

[4/4] Utilisation et droits

Droits essentiels

```
-- Autoriser l'utilisation du schéma et la création d'objets
GRANT USAGE, CREATE ON SCHEMA app TO employes;

-- Droits sur les objets existants (tables)
GRANT SELECT, INSERT, UPDATE, DELETE
  ON ALL TABLES IN SCHEMA app TO employes;

-- (Optionnel) Droits sur les séquences existantes
GRANT USAGE ON ALL SEQUENCES IN SCHEMA app TO employes;
```

Bonnes pratiques

- Toujours qualifier les objets : `schema.table`.
- Restreindre les droits du schéma `public`.
- Créer un schéma par domaine applicatif.
- (Optionnel) Propager les droits aux objets futurs : `ALTER DEFAULT PRIVILEGES`.

Travailler sur une base : propriétaire ou privilèges

Pour travailler sur une base, deux situations se présentent :

- **Rôle propriétaire** — il dispose de tous les droits sur la base.
- **Rôle non propriétaire** — il doit recevoir les privilèges nécessaires.

Exemple : accorder les privilèges sur une base

```
-- Accorder tous les droits sur la base "pizzerias" à rôle "vendeur"
GRANT ALL PRIVILEGES ON DATABASE pizzerias TO vendeur;
```

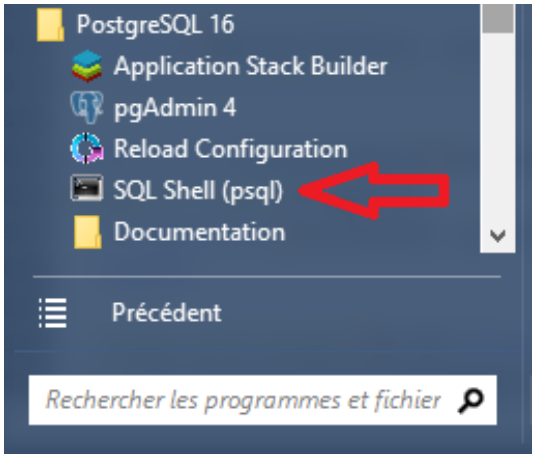
Niveaux de privilèges

- *Schéma* — USAGE, CREATE (création d'objets).
- *Tables* — SELECT, INSERT, UPDATE, DELETE.
- *Séquences* — USAGE, SELECT, UPDATE (pour SERIAL/IDENTITY).
- *Fonctions / procédures* — EXECUTE.
- *Privilèges par défaut* — `ALTER DEFAULT PRIVILEGES`.
- *Rôles-groupes* — `GRANT groupe TO utilisateur`.

Connexion au serveur PostgreSQL

Pour se connecter à PostgreSQL :

- **Méthode 1 :** lancer le **SQL Shell (psql)** depuis son icône.



- **Méthode 2 :** ouvrir un terminal et exécuter la commande suivante :

```
C:\PostgreSQL\16\bin\psql.exe -h localhost -U postgres -d postgres -
    p 5432
```

Si le chemin de **psql** est dans le **PATH** :

```
C:\> psql -h localhost -U postgres -d postgres -p 5432
```

Commandes internes de psql

Les commandes internes de psql permettent d'administrer le serveur sans passer par le langage SQL. Elles commencent toujours par une barre oblique inversée \.

Commandes principales

```
\l              -- Lister les bases de données
\c dbname       -- Se connecter à une base
\dt             -- Lister les tables
\d table        -- Décrire une table
\du             -- Lister les rôles et utilisateurs
\dn             -- Lister les schémas
\conninfo       -- Informations sur la connexion active
\?              -- Aide sur les commandes psql
\h commande     -- Aide sur une commande SQL (ex: \h SELECT)
\i fichier      -- Exécuter un script SQL
\o fichier      -- Rediriger la sortie vers un fichier
\timing         -- Activer/désactiver la mesure du temps
\password       -- Modifier le mot de passe d'un rôle

\q              -- Quitter psql
```

Base exemple pédagogique

But : disposer d'une base simple prise comme exemple pour illustrer la création, la gestion des droits et les requêtes SQL dans PostgreSQL.

Base utilisée :

- `pizzerias` — base exemple pour l'initiation : ventes de pizzas (tables simples, clés étrangères, cascades).

Ressources

Scripts SQL, PDF des TPs et supports disponibles sur : `bda.al-moualime.com`