

SGBD

BTS DAKHLA

DAHAR RACHID

Rappel — qu'est-ce qu'une base de données ?

Une base de donnée est un ensemble structuré et organisé de données enregistrées sur des supports accessible par l'ordinateur

Exemple concret : la liste papier d'un club où chaque ligne contient nom, téléphone, date d'inscription — la base remplace ce classeur papier par un stockage structuré sur ordinateur.

Table = un "tableau" dans la base, avec des colonnes (types d'informations) et des lignes (chaque enregistrement).

But : éviter les fichiers dispersés, les doublons, les erreurs, faciliter les recherches et la sécurité

Rappel — À quoi sert un SGBD ?

SGBD = logiciel qui gère une ou plusieurs bases de données : il organise, sécurise, et répond aux demandes (lire, écrire, modifier).

Utilités principales :

1. **Stocker** des données de façon organisée (tables, index).
2. **Garantir l'intégrité** : empêcher les erreurs logiques (ex : ne pas permettre de prêt pour un adhérent qui n'existe pas).
3. **Gérer la concurrence** : plusieurs personnes peuvent accéder ou modifier les mêmes données sans tout casser.
4. **Sécuriser** l'accès (qui peut lire, qui peut écrire).
5. **Sauvegarder / restaurer** : pouvoir revenir en arrière en cas de panne.
6. **Optimiser** les recherches (index, caches) pour que les requêtes restent rapides même avec beaucoup de données.

Rappel — À quoi sert un SGBD ?

SGBD = logiciel qui gère une ou plusieurs bases de données : il organise, sécurise, et répond aux demandes (lire, écrire, modifier).

Utilités principales :

1. **Stocker** des données de façon organisée (tables, index).
2. **Garantir l'intégrité** : empêcher les erreurs logiques (ex : ne pas permettre de prêt pour un adhérent qui n'existe pas).
3. **Gérer la concurrence** : plusieurs personnes peuvent accéder ou modifier les mêmes données sans tout casser.
4. **Sécuriser** l'accès (qui peut lire, qui peut écrire).
5. **Sauvegarder / restaurer** : pouvoir revenir en arrière en cas de panne.
6. **Optimiser** les recherches (index, caches) pour que les requêtes restent rapides même avec beaucoup de données.

Exemples SGBD

ORACLE®



PostgreSQL

Pourquoi choisir PostgreSQL plutôt qu'un autre SGBD ?

Quelques SGBD connus : MySQL/MariaDB, SQLite, PostgreSQL, Oracle, SQL Server. PostgreSQL

— **points forts (en termes simples) :**

- **Open-source :** gratuit et modifiable, large communauté.
- **Robuste et fidèle au standard SQL :** respecte bien les règles du langage, utile pour apprendre et pour des projets sérieux.
- **Fonctionnalités avancées :** types de données riches (JSON, géométrie...), possibilités d'extensions, et gestion solide des transactions (opérations regroupées).
- **Bon pour les requêtes complexes** (rapports, jointures nombreuses) et les applications professionnelles.

Comparaison rapide :

- **MySQL / MariaDB** : très répandu pour sites web, configuration souvent plus simple ; PostgreSQL est souvent choisi si vous avez des besoins complexes (intégrité, types avancés, requêtes lourdes).
- **SQLite** : fichier léger, idéal pour applications simples ou tests, mais pas pour multi-utilisateurs concurrents.
- **Oracle / SQL Server** : solutions commerciales puissantes ; coûteuses. PostgreSQL est une alternative gratuite très solide.

Conclusion : PostgreSQL est un excellent choix pédagogique (apprendre des concepts solides) et industriel (projets réels).

Qui sont les utilisateurs d'un SGBD ?

- **DBA (Administrateur de base de données)** : gère sauvegardes, performance, sécurité.
- **Développeurs** : conçoivent l'application qui dialogue avec la base.
- **Analystes / Data scientists** : extraient et analysent les données.
- **Utilisateurs finaux** : utilisent l'interface (site, application) qui fait des requêtes vers la base.
- **Support / Opérations** : surveillent le bon fonctionnement.

Architecture de PostgreSQL

- **Postmaster / processus** : le "chef" qui reçoit les connexions ; PostgreSQL ouvre un processus pour chaque connexion.

Analogie : un bureau d'accueil qui crée un guichet pour chaque visiteur.

- **Shared buffers (mémoire partagée)** : zone mémoire commune pour garder des morceaux de données récemment utilisés (plus rapide que disque).

Analogie : photocopies en libre-service pour éviter d'aller à la réserve.

- **WAL (Write-Ahead Log)** : avant d'appliquer une modification, PostgreSQL écrit d'abord dans un journal (WAL). Si l'ordinateur plante, on relit ce journal pour revenir à un état cohérent.

Analogie : noter dans un carnet ce qu'on va faire avant de modifier le registre officiel.

Architecture de PostgreSQL

- **MVCC (Multi-Version Concurrency Control)** : méthode qui permet à plusieurs personnes de lire et écrire sans se bloquer. PostgreSQL garde plusieurs versions d'une même ligne tant que nécessaire.

Analogie : copies successives d'un document — chacun voit la version qui lui convient.

- **Autovacuum (nettoyage automatique)** : PostgreSQL nettoie périodiquement les anciennes versions pour libérer de l'espace et maintenir les performances.
- **Checkpoints** : points où l'état est solidifié sur disque pour limiter le travail à refaire en cas de crash.

SQL : Structured Query Language

En SQL (langage pour parler au SGBD), on distingue souvent plusieurs sous-ensembles.

- **LDD / DDL (Langage de Définition de Données / Data Definition Language)**

→ commandes qui **définissent** la structure (créer ou modifier tables, index, vues, bases). Exemples : CREATE, ALTER, DROP.

- **LMD / DML (Langage de Manipulation de Données / Data Manipulation Language)**

→ commandes pour **manipuler** les données (ajouter, lire, modifier, supprimer). Exemples : INSERT, SELECT, UPDATE, DELETE.

- **LCD / DCL (Langage de Contrôle des Données / Data Control Language)**

→ commandes pour la **sécurité** et les **permissions**. Exemples : GRANT, REVOKE.

- **LCT / TCL (Langage de Contrôle des Transactions / Transaction Control Language)**

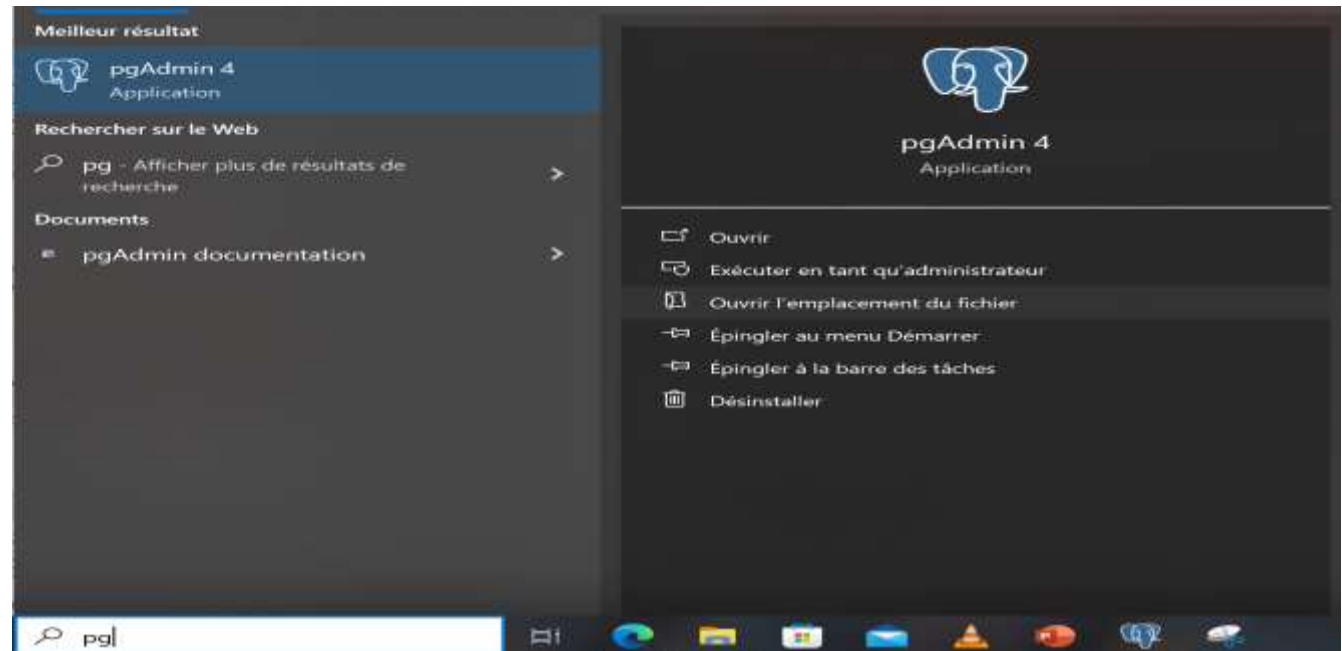
→ commandes pour gérer les transactions (groupes d'opérations). Exemples : BEGIN, COMMIT, ROLLBACK.

Installation de PostgreSQL

1. Téléchargez l'installateur PostgreSQL (EnterpriseDB).
2. Lancez l'installateur .exe. Suivez les écrans :
 - choisissez le dossier d'installation,
 - le port (laisser 5432),
 - créez le mot de passe pour postgres,
 - choisissez le locale.
3. L'installateur propose souvent d'installer pgAdmin (outil graphique) — acceptez si vous voulez.
4. À la fin, le service PostgreSQL est généralement démarré automatiquement.

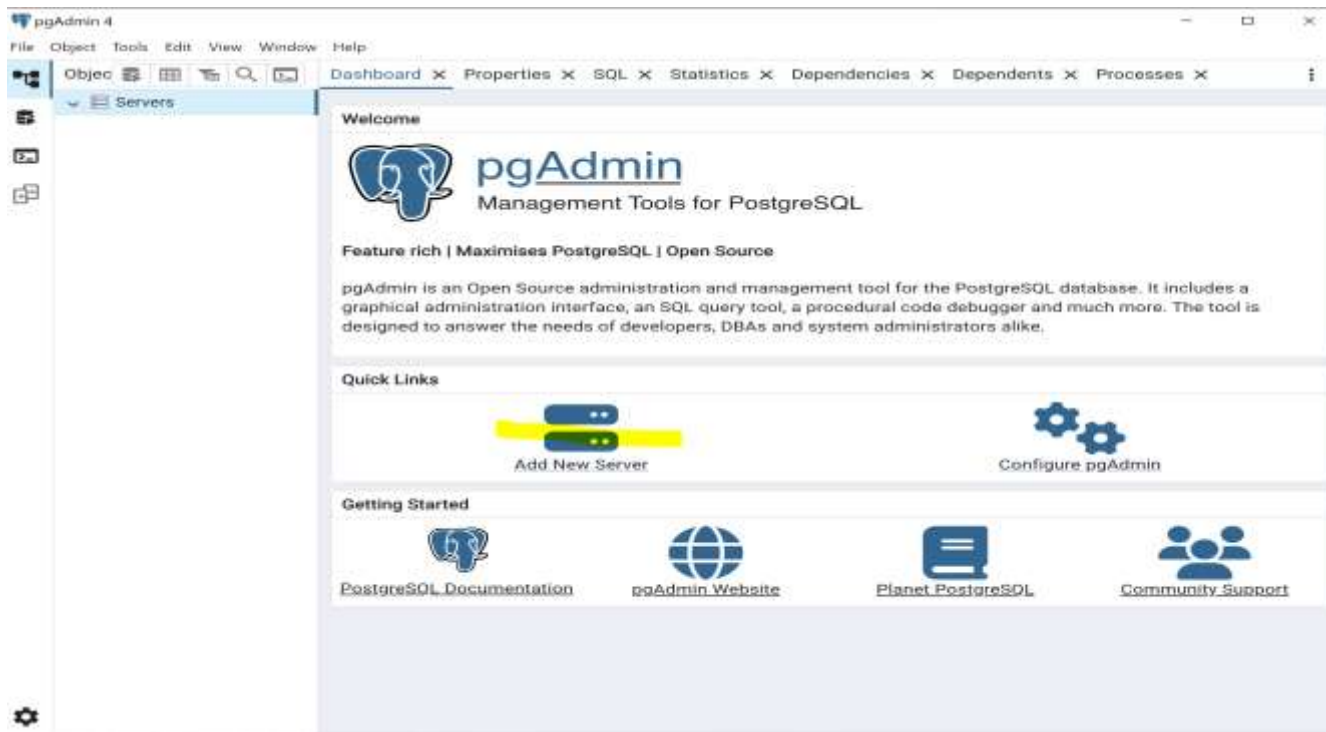
Première configuration après l'installation

1. Étape 1 : Lancer pgAdminChercher pgAdmin 4 dans le menu démarrer et l'ouvrir.La première fois



Première configuration après l'installation

1. Étape 2 : Ajouter un serveur PostgreSQL Une fois pgAdmin ouvert clique sur **Add New Server**

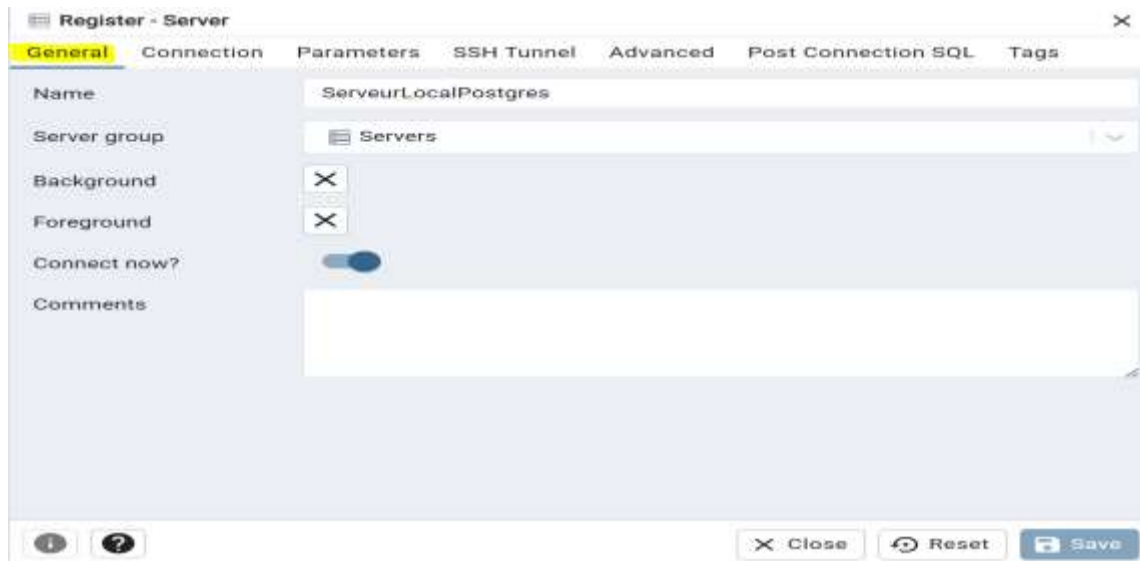


Première configuration après l'installation

1. fenêtre apparaît avec plusieurs onglets :

Onglet 1 : General

Name : Nom logique pour le serveur **Exemple** : ServeurLocalPostgres



The screenshot shows a 'Register - Server' dialog box with the 'General' tab selected. The dialog has a title bar with a close button (X). Below the title bar are tabs: 'General' (highlighted), 'Connection', 'Parameters', 'SSH Tunnel', 'Advanced', 'Post Connection SQL', and 'Tags'. The 'General' tab contains the following fields and controls:

- Name**: A text input field containing 'ServeurLocalPostgres'.
- Server group**: A dropdown menu showing 'Servers'.
- Background**: A checkbox with an 'X' icon, currently unchecked.
- Foreground**: A checkbox with an 'X' icon, currently unchecked.
- Connect now?**: A toggle switch, currently turned on (blue).
- Comments**: A large text area for additional notes.

At the bottom of the dialog, there are three buttons: 'Close' (with an X icon), 'Reset' (with a circular arrow icon), and 'Save' (with a floppy disk icon).

Première configuration après l'installation

Onglet 2 : Connection

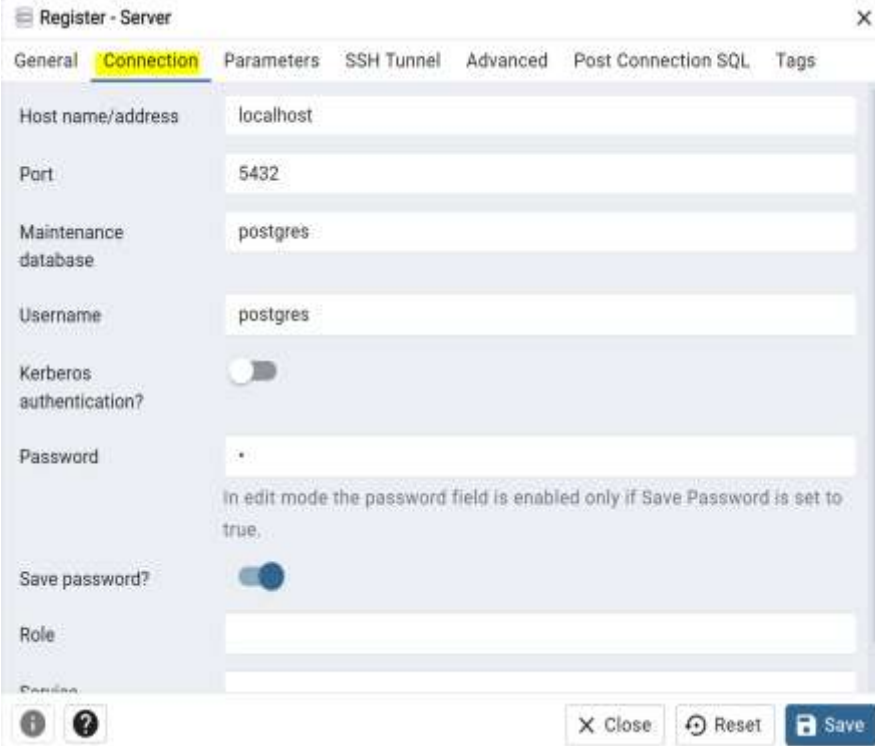
Host name/address : Si PostgreSQL est sur votre PC → localhost

Port : Par défaut → 5432

Maintenance database : postgres (base par défaut)

Username : postgres (superutilisateur créé à l'installation)

Password : le mot de passe choisi à l'installation (exemple : admin123) On peut cocher Save password pour ne pas le retaper à chaque fois. Puis cliquer **Save**.



The screenshot shows the 'Register - Server' configuration window with the 'Connection' tab selected. The fields are filled with the following values:

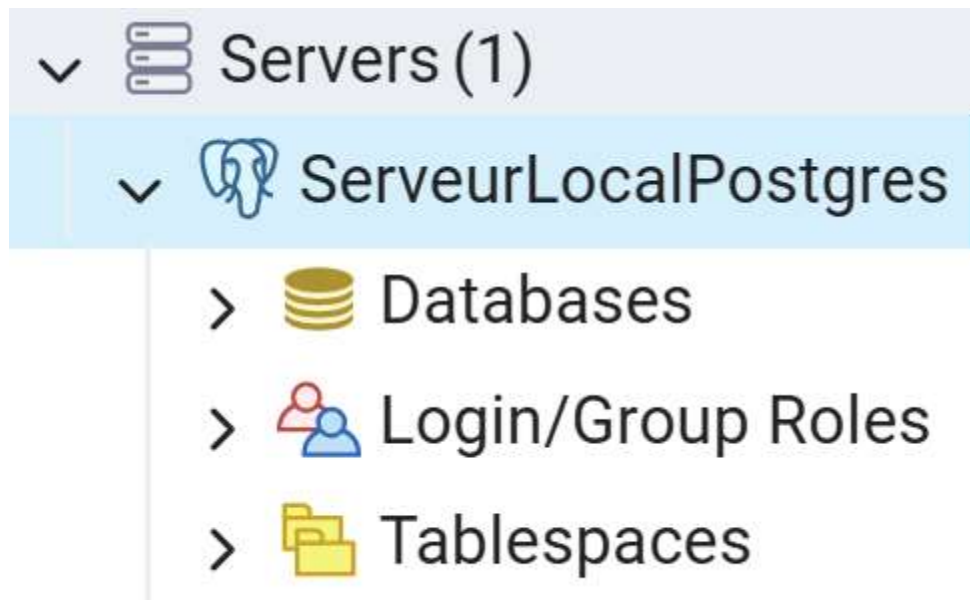
- Host name/address: localhost
- Port: 5432
- Maintenance database: postgres
- Username: postgres
- Kerberos authentication?: ☐
- Password: [masked]
- Save password?: ☒
- Role: [empty]

At the bottom right, there are three buttons: 'Close', 'Reset', and 'Save'.

Première configuration après l'installation

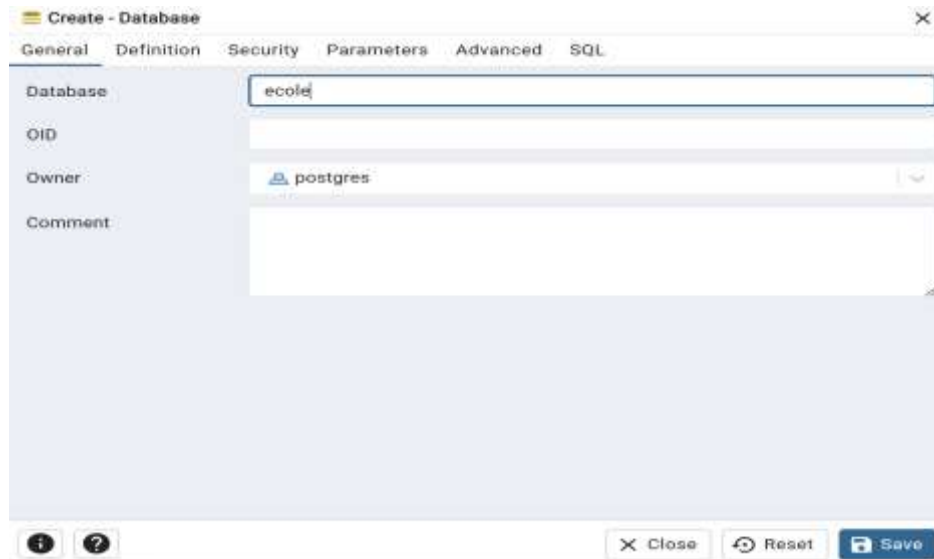
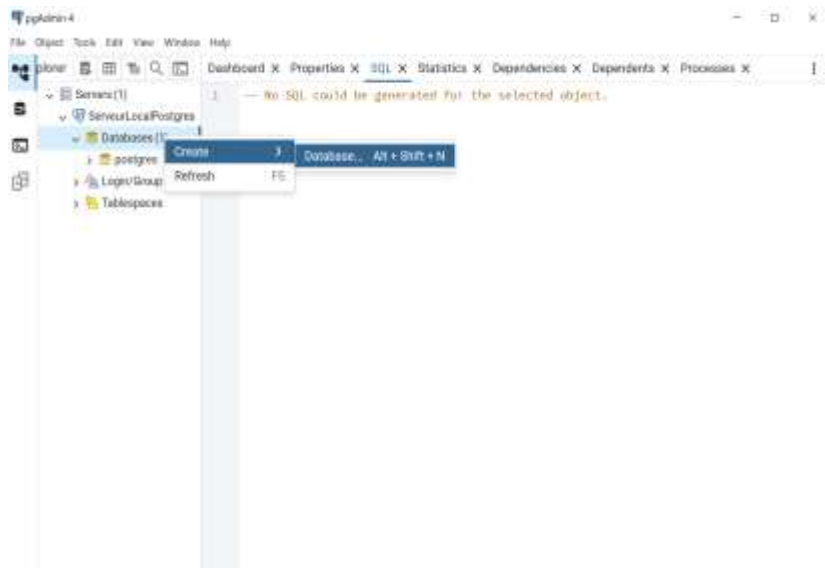
Étape 3 : Vérification

Si tout est correct, le serveur s'affiche



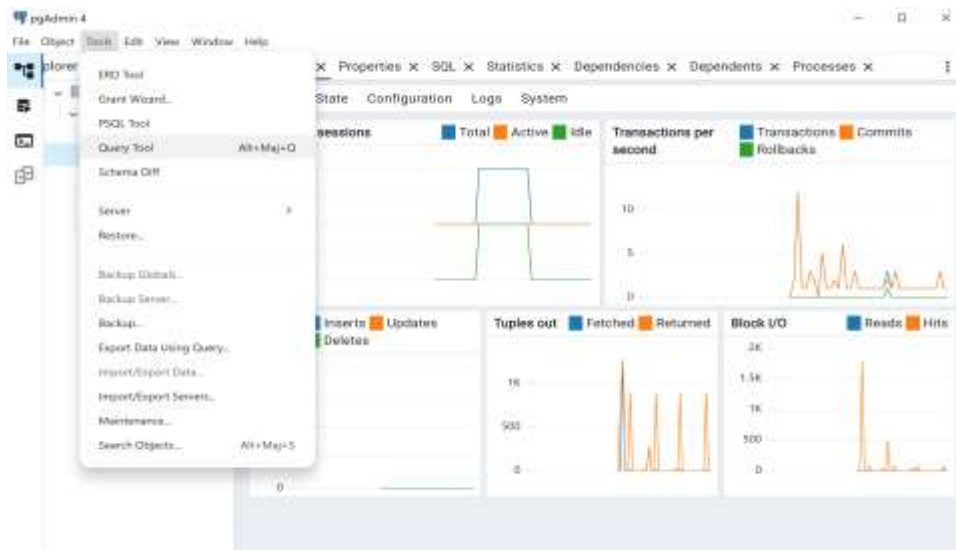
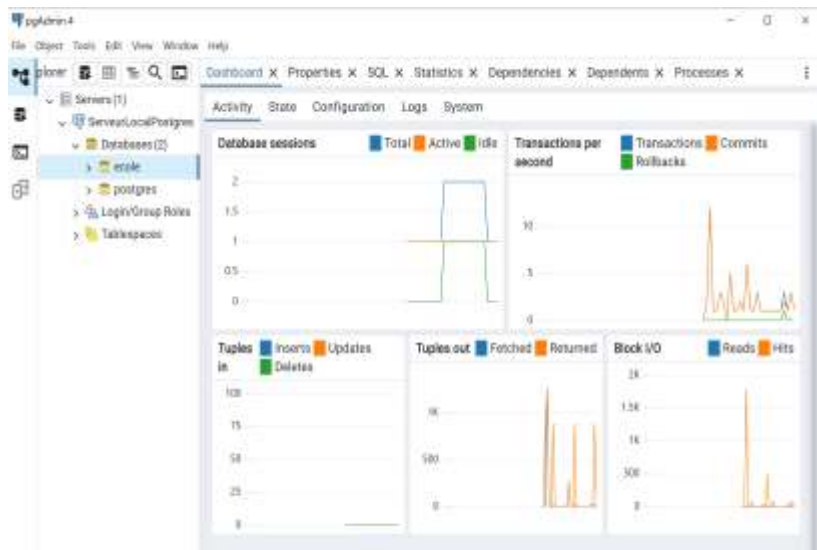
éditeur de requêtes

1. Créer une base de données (si nécessaire)
 - Clique droit sur Databases → Create → Database.
 - Donne un nom à ta base (ex. : ecole) → clique sur Save.



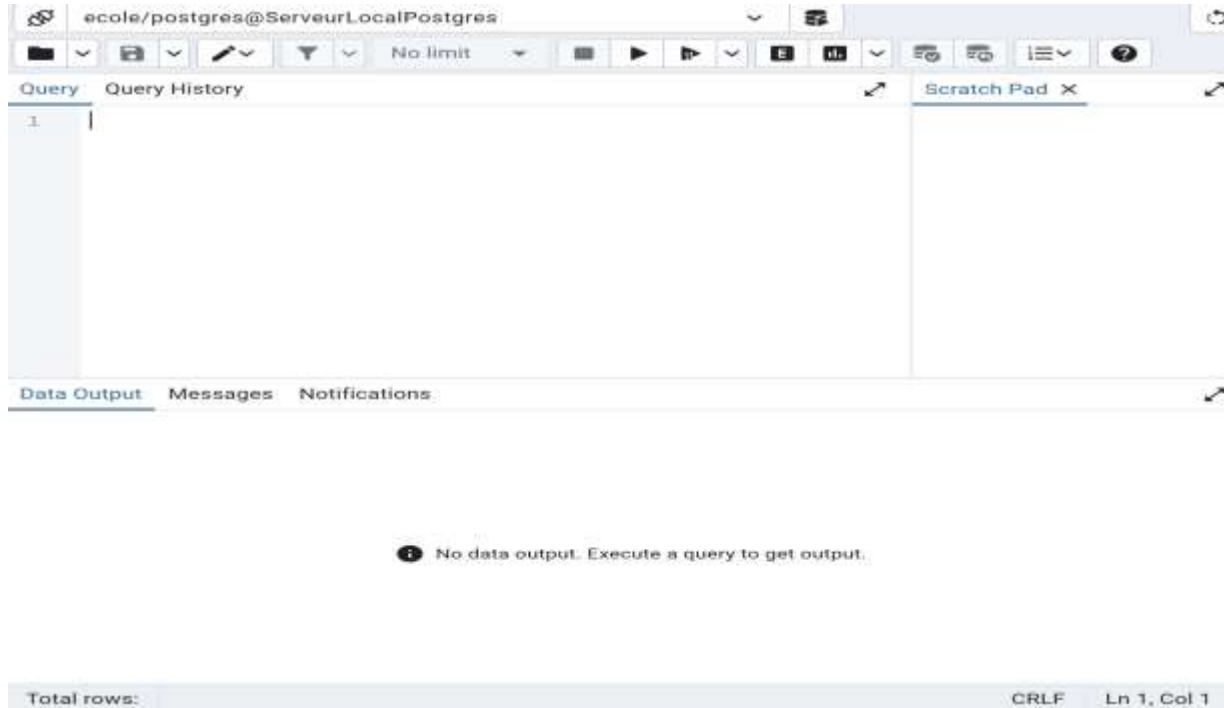
éditeur de requêtes

1. Dans la barre de gauche, clique sur le nom de ta base (ecole).
2. En haut, clique sur Tools → Query Tool.
3. Une fenêtre s'ouvre : c'est ici que tu peux écrire tes requêtes SQL.



éditeur de requêtes

1. Une fenêtre s'ouvre : c'est ici que tu peux écrire tes requêtes SQL.



Qu'est-ce que le LLD en SQL ?

Le **Langage de Définition de Données (LLD)** est la partie du SQL qui sert à **définir la structure** d'une base de données.

Avec le LLD, on peut :

- Créer une base de données
- Créer des schémas (sous-espaces logiques)
- Créer des tables et définir leurs colonnes
- Modifier une structure (ajouter/supprimer colonnes)
- Supprimer une table, une base, un schéma

En PostgreSQL, le LLD comprend les instructions comme :

- **CREATE** → créer
- **ALTER** → modifier
- **DROP** → supprimer
- **TRUNCATE** → vider une table

Qu'est-ce qu'un Tablespace ?

Un **tablespace** est un **emplacement physique** sur le disque où PostgreSQL stocke les fichiers d'une base de données.

Sans tablespace, PostgreSQL ne saurait pas où stocker tes données.

Quand tu installes PostgreSQL, deux tablespaces sont créés automatiquement :

- **pg_default** Tablespace utilisé par défaut pour les données
- **pg_global** Tablespace pour les données globales (système)

Si tu ne précises pas de tablespace lors de la création d'une base, pg_default est utilisé automatiquement

Création d'un nouveau tablespace

```
CREATE TABLESPACE mon_espace LOCATION '/chemin/vers/dossier'
```

Création de Base de données

Voici d'abord la façon la plus courte pour créer une base :

```
CREATE DATABASE ma_base;
```

Crée ma_base avec toutes les options par défaut (propriétaire = utilisateur courant, encodage/locale hérités de template1, tablespace = pg_default, etc.)

1) OWNER — définir le propriétaire

```
CREATE DATABASE ma_base OWNER = mon_user;
```

Rôle : qui possède la base (crédits pour créer/administrer).

Défaut si omis : l'utilisateur qui exécute la commande.

2) ENCODING — jeu de caractères

C'est le jeu de caractères utilisé pour stocker les données dans la base.

Exemple :

- **UTF8** permet de stocker des textes avec toutes les langues (français, arabe, chinois...).
- Si on choisit **SQL_ASCII**, on ne pourra pas gérer correctement les accents ou certaines langues.
- **Défaut si omis** : hérite de template1 (souvent UTF8).

CREATE DATABASE ma_base **ENCODING** = 'UTF8'

➡ On peut enregistrer : Élève, Étudiant, مدرسة, 学生

3) LC_COLLATE — ordre de tri

```
CREATE DATABASE ma_base LC_COLLATE = 'fr_FR.UTF-8';
```

Rôle : règle de tri des textes (ex. ordre alphabétique français).

Défaut si omis : hérite de template1.

4) LC_CTYPE — classification de caractères (maj/min)

```
CREATE DATABASE ma_base LC_CTYPE = 'fr_FR.UTF-8';
```

Rôle : comment PostgreSQL traite majuscules/minuscules et catégories de caractères (utile pour UPPER/LOWER).

Défaut si omis : hérite de template1.

5) TABLESPACE — emplacement disque

```
CREATE DATABASE ma_base TABLESPACE = ts_ecole;
```

Rôle : tablespace physique où seront stockés fichiers de la base.

Défaut si omis : pg_default.

Important : ts_ecole doit exister (créé auparavant par CREATE TABLESPACE ... LOCATION 'chemin').

6) CONNECTION LIMIT — limite de connexions

```
CREATE DATABASE ma_base CONNECTION LIMIT = 50;
```

Rôle : nombre maximal de connexions simultanées à la base.

Défaut si omis : -1 (illimité).

7) ALLOW_CONNECTIONS — autoriser les connexions

```
CREATE DATABASE ma_base ALLOW_CONNECTIONS = FALSE;
```

Rôle : TRUE = autorise les connexions, FALSE = bloque toute connexion (utile pour maintenance).

Défaut si omis : TRUE.

8) IS_TEMPLATE — rendre la base réutilisable comme modèle

```
CREATE DATABASE ma_base IS_TEMPLATE = TRUE;
```

Rôle : si TRUE, la base peut servir de modèle pour CREATE DATABASE ... TEMPLATE = ma_base.

Défaut si omis : FALSE

3) LC_COLLATE — ordre de tri

```
CREATE DATABASE ma_base LC_COLLATE = 'fr_FR.UTF-8';
```

Rôle : règle de tri des textes (ex. ordre alphabétique français).

Défaut si omis : hérite de template1.

4) LC_CTYPE — classification de caractères (maj/min)

```
CREATE DATABASE ma_base LC_CTYPE = 'fr_FR.UTF-8';
```

Rôle : comment PostgreSQL traite majuscules/minuscules et catégories de caractères (utile pour UPPER/LOWER).

Défaut si omis : hérite de template1.

Création d'une base de donnée

```
CREATE DATABASE ecole
WITH
    OWNER = directeur_ecole
    ENCODING = 'UTF8'
    LC_COLLATE = 'fr_FR.UTF-8'
    LC_CTYPE = 'fr_FR.UTF-8'
    TABLESPACE = ts_ecole
    CONNECTION LIMIT = 100
    ALLOW_CONNECTIONS = TRUE
    IS_TEMPLATE = FALSE;
```

Pour connaître les valeurs héritées :

```
SELECT encoding, datcollate, datatype FROM pg_database WHERE datname='template1';
```

LE LANGAGE SQL-LDD : CRÉATION D'UNE TABLE

- ❑ La commande CREATE TABLE sert à spécifier une nouvelle relation en lui attribuant un **nom**, des **attributs** et des **contraintes**
- ❑ Les attributs doivent être en premier. Chaque attribut doit avoir un **nom** , un **type de données** (pour indiquer le domaine auquel appartiennent ses valeurs) et des **contraintes** éventuelles (NOT NULL par exemple)
- ❑ Les **contraintes de clé** , **d'intégrité de l'entité** et **d'intégrité référentielle** peuvent être spécifiées dans l'instruction **CREATE TABLE** après la déclaration des attributs ou être ajoutées plus tard à l'aide de la commande **ALTER TABLE**

LE LANGAGE SQL-LDD : CRÉATION D'UNE TABLE

```
CREATE TABLE nom_table(  
  
    nom_colonne      type_donnee_colonne [definition_contrainte_colonne],  
  
    nom_colonne      type_donnee_colonne [definition_contrainte_colonne],  
  
    ....  
  
    [definition_contrainte_table],  
  
    ....  
  
    [definition_contrainte_table]  
  
);
```


TYPE DE DONNÉES ET DOMAINES DES ATTRIBUTS

❑ **SQL** offre divers **types de données** de base , dans la déclaration d'une colonne d'une table . On citera les principaux existant en **PostgreSQL**:

- **SMALLINT** : entier signés courts (ex 16 bits)
- **INTEGER (ou INT)** : entier signés longs (ex 32 bits)
- **NUMERIC(p, q) et DECIMAL(p, q) sont strictement identiques.**
 - **p** = nombre total de chiffres (avant et après la virgule)
 - **q** = nombre de chiffres après la virgule
 - **Si q est omis**, il vaut par défaut 0 → pas de partie décimale autorisée.
- **REAL ou DOUBLE PRECISION** : nombre en virgules flottante

TYPE DE DONNÉES ET DOMAINES DES ATTRIBUTS

- **CHAR (n)** : chaîne de longueur fixe de n caractère , La chaîne est complétée par des espace si elle est plus petite que la taille déclarée
- **VARCHAR (n)** : chaîne de longueur variables d'au plus n caractères
- **DATE** : dates (année , mois et jours)
- **TIME** : instants (heure , minute , seconde)
- **TIMESTAMP** : date + temps exemple: '2024-09-25 19:15:45'.

En plus de ces type de base , il est possible de définir des types :

```
CREATE DOMAIN montant AS NUMERIC(10,2) CHECK (VALUE >= 0);
```

CONTRAINTES D'INTÉGRITÉ

❑ Les contraintes de base (contraintes de colonnes) permettant de spécifier différentes contraintes portant sur un seul attribut, lors de la création d'une table . Parmi celle-ci nous avons :

- **Contraintes sur les attributs** et de leurs **valeurs par défaut** (NOT NULL , DEFAULT , CHECK)
- **Contrainte de clé (unicité de l'attribut)** : UNIQUE ou PRIMARY KEY
- **Contrainte référentielle** : FOREIGN KEY

RÈGLE RELATIVES AUX CONTRAINTES

☐ Vous pouvez créer une contrainte à différents instants:

- Lors de la création de la table
- Après la création de la table

☐ Définissez une contrainte au **niveau colonne** ou **au niveau table**.

Niveau Colonne :

```
CREATE TABLE Products (  
    ProductID INT CONSTRAINT pk_products_pid PRIMARY KEY,  
    ProductName VARCHAR(25)  
);
```

Niveau Table :

```
CREATE TABLE Products (  
    ProductID INT,  
    ProductName VARCHAR(25),  
    CONSTRAINT pk_products_pid PRIMARY KEY(ProductID)  
);
```

CONTRAINTES SUR LES ATTRIBUTS ET DE LEURS VALEURS PAR DÉFAUT

Valeur nulle impossible :

- **SQL** permet des valeurs NULL pour les attributs , et il est possible de spécifier une contrainte NOT NULL pour interdire cette valeur.
- Cette contrainte est spécifier implicitement pour les attributs qui font partie de la **clé primaire**, mais peut être spécifiée pour n'importe quel autre attribut dont il est obligatoire que les valeur ne soient pas NULL

EXEMPLE

```
CREATE TYPE montant FROM decimal(6,2)
```

```
CREATE TABLE Employe (  
    Matricule INT NOT NULL,  
    Nom VARCHAR(20) NOT NULL,  
    Prenom VARCHAR(20),  
    Sexe CHAR(1),  
    Ville VARCHAR(20),  
    Email VARCHAR(25) ,  
    Salaire montant -- il faut créer le type montant  
);
```

CONTRAINTES SUR LES ATTRIBUTS ET DE LEURS VALEURS PAR DÉFAUT

Valeur par Défaut :

- Il est possible de définir la valeur par défaut d'un attribut en ajoutant la clause **DEFAULT** <valeur> à sa définition.
- La valeur par défaut est alors incluse dans tout nouveau tuple pour lequel il n'a pas été explicitement fourni de valeur.
- En absence de clause défaut, la valeur par défaut est NULL pour les attribut sur lesquels ne porte pas de contrainte NOT NULL.

EXEMPLE

```
CREATE TABLE Employe (  
    Matricule INT NOT NULL,  
    Nom VARCHAR(20) NOT NULL,  
    Prenom VARCHAR(20) ,  
    Sexe CHAR(1) DEFAULT 'M',  
    Ville VARCHAR(20) DEFAULT 'DAKHLA',  
    Email VARCHAR(25) ,  
    Salaire montant -- il faut créer le type montant  
);
```

```
CREATE TABLE Employe (  
    Matricule INT NOT NULL ,  
    Nom VARCHAR(20) NOT NULL,  
    Prenom VARCHAR(20) ,  
    Sexe CHAR(1) Constraint DF_Sexe DEFAULT 'M',  
    Ville VARCHAR(20) Constraint DF_Ville DEFAULT 'DAKHLA',  
    Email VARCHAR(25),  
    Salaire montant -- il faut créer le type montant  
);
```


CONTRAINTES SUR LES ATTRIBUTS ET DE LEURS VALEURS PAR DÉFAUT

La contrainte CHECK impose un domaine de valeurs ou une condition simple ou complexe entre colonnes.

CHECK (note BETWEEN 0 AND 20),

CHECK (grade='Copilote' OR grade='Commandant')

EXEMPLE

```
CREATE TABLE Employe (  
    Matricule INT NOT NULL,  
    Nom VARCHAR(20) NOT NULL,  
    Prenom VARCHAR(20) ,  
    Sexe CHAR(1) DEFAULT 'M' CHECK (Sexe IN ('M', 'F')),  
    Ville VARCHAR(20) DEFAULT 'DAKHLA',  
    Email VARCHAR(25) ,  
    Salaire montant CHECK (Salaire BETWEEN 0 AND 10000)  
);
```

```
CREATE TABLE Employe (  
    Matricule INT NOT NULL,  
    Nom VARCHAR(20) NOT NULL,  
    Prenom VARCHAR(20) ,  
    Sexe CHAR(1) Constraint DF_Sexe DEFAULT 'M' Constraint CK_Sexe CHECK (Sexe IN  
        ('M', 'F')),  
    Ville VARCHAR(20) Constraint DF_Ville DEFAULT 'DAKHLA',  
    Email VARCHAR(25),  
    Salaire montant Constraint CK_Salaire CHECK (Salaire BETWEEN 0 AND 10000)  
);
```

CONTRAINTES DE CLÉ

La Clause **PRIMARY KEY** désigne le ou les attributs qui forment la clé primaire d'une relation lors de la création d'une table

```
CREATE TABLE Employe (  
    Matricule INT PRIMARY KEY,  
    Nom VARCHAR(20) NOT NULL,  
    Prenom VARCHAR(20) ,  
    Sexe CHAR(1) DEFAULT 'M' CHECK (Sexe IN ('M', 'F')),  
    Ville VARCHAR(20) DEFAULT 'DAKHLA',  
    Email VARCHAR(25),  
    Salaire montant CHECK (Salaire BETWEEN 0 AND 10000)  
    -- il faut créer le type montant  
);
```

SERIAL

Type serial en PostgreSQL qui permet de créer une colonne auto-incrémentée automatiquement à chaque insertion. Il définit implicitement un entier et simplifie l'ajout de nouvelles valeurs.

Le type serial convient surtout pour les tables ayant comme clé primaire un numéro séquentiel .

```
CREATE TABLE Carburant (  
    IdType SERIAL PRIMARY KEY,  
    TypeCarb VARCHAR(20) NOT NULL,  
);
```

CONTRAINTES DE CLÉ

La clause **UNIQUE** Indique que les valeurs saisies pour les colonnes (attributs) doivent être uniques, ce qui veut dire pas de doublons.

Exemple: Le numéro d'assurance sociale, le numéro d'un permis de conduire, un courriel.

```
CREATE TABLE Employe (  
    Matricule INT PRIMARY KEY,  
    Nom VARCHAR(20) NOT NULL,  
    Prenom VARCHAR(20) ,  
    Sexe CHAR(1) DEFAULT 'M' CHECK (Sexe IN ('M', 'F')),  
    Ville VARCHAR(20) DEFAULT 'DAKHLA',  
    Email VARCHAR(25) UNIQUE,  
    Salaire montant CHECK (Salaire BETWEEN 0 AND 10000)  
    -- il faut créer le type montant  
);
```

CONTRAINTE D'INTÉGRITÉ RÉFÉRENTIELLE

- On parle d'intégrité référentielle lorsque les valeurs d'une ou plusieurs colonnes d'une table sont déterminées ou font référence à des valeurs d'une colonne d'une autre table.
- Dans notre exemple ci-après, les valeurs de la colonne codeequipe de la table Joueurs font référence aux valeurs de la colonne codeequipe de la table Equipes
- La table Joueurs ne contient aucun codeequipe qui n'est pas dans la table Equipes.

Table joueurs

NUMJOUEUR	NOM	PRENOM	CODEEQUIPE
1	PRICE	CAREY	MTL
2	MARKOV	ANDRÉ	MTL
3	SUBBAN	KARL	MTL
4	PATIORETTY	MAX	MTL
10	HAMOND	ANDREW	OTT
6	STONE	MARC	OTT
9	TURIS	KYLE	OTT
7	GALLAGHER	BRANDON	MTL
8	TANGUAY	ALEX	AVL
11	THOMAS	BIL	AVL

Table Equipes

CODEEQUIPE	NOMEQUIPE	VILLE	NBCOUPES
1 MTL	LES CANADIENS DE MONTRÉAL	MONTRÉAL	24
2 TOR	LES MAPLE LEAFS	TORONTO	22
3 OTT	LES SÉNATEURS	OTTAWA	4
4 AVL	LES AVALANCHES	COLORADO	2
5 VAN	LES CANUKS	VANCOUVER	1
6 BRU	LES BRUNS DE BOSTON	BOSTON	13

référence

La table EQUIPES doit être créée en premier

```
CREATE table EQUIPES(  
    codeequipe CHAR(3) CONSTRAINT pk_equipe PRIMARY KEY,  
    nomequipe VARCHAR(50) NOT NULL,  
    Ville VARCHAR(40),  
    nbcoupes NUMERIC(2,0) CONSTRAINT ck_nbcoupe CHECK (nbcoupes > =0)  
);
```

On crée la table JOUEURS après

```
CREATE TABLE Joueurs(  
    numjoueur NUMERIC(3,0) CONSTRAINT pk_joueurs PRIMARY KEY,  
    nom VARCHAR(30) NOT NULL,  
    prenom VARCHAR(30),  
    codeequipe CHAR(3),  
    CONSTRAINT fk_Joueurs_equipes FOREIGN KEY (codeequipe) REFERENCES  
        equipes(codeequipe)  
);
```

- Les attributs codeequipe des deux tables sont de même type et de même longueur: CHAR(3). C'est une obligation.
- Le mot réservé REFERENCES indique la colonne (attribut) référencée de la table de la clé primaire. C'est pourquoi la table de la clé primaire doit être créée en premier. On ne peut pas référencer quelque chose qui n'existe pas.

CONTRAINTE FOREIGN KEY : MOTS CLÉS : ON DELETE

- **ON DELETE CASCADE** : supprime les lignes dépendantes dans la table enfant lorsqu'une ligne de la table parent est supprimée.
- **ON DELETE SET NULL**: convertit les valeurs des clés étrangères dépendantes en valeurs NULL. Cette contrainte ne peut être exécutée que si toutes les colonnes de clé étrangère de la table cible acceptent des valeurs NULL.
- **SET DEFAULT** place la valeur par défaut (qui suit ce paramètre) dans la ligne de la table étrangère en cas d'effacement d'une valeur correspondant à la clé

On crée la table JOUEURS après

```
CREATE TABLE Joueurs(  
  numjoueur NUMERIC(3,0) CONSTRAINT pk_joueurs PRIMARY KEY,  
  nom VARCHAR(30) NOT NULL,  
  prenom VARCHAR(30),  
  codeequipe CHAR(3),  
  CONSTRAINT fk_Joueurs_equipes FOREIGN KEY (codeequipe) REFERENCES equipes(codeequipe)  
    ON DELETE CASCADE  
);
```


CONTRAÎNTE FOREIGN KEY : MOTS CLÉS : ON UPDATE

- **ON UPDATE CASCADE** : Les valeurs associées dans la table enfant seront également mises à jour.
- **ON UPDATE SET NULL**: convertit les valeurs des clés étrangères dépendantes en valeurs NULL. Cette contrainte ne peut être exécutée que si toutes les colonnes de clé étrangère de la table cible acceptent des valeurs NULL.
- **ON UPDATE SET DEFAULT** Les valeurs associées dans la table enfant seront définies par la valeur par défaut spécifiée dans la définition de la colonne.

On crée la table JOUEURS après

```
CREATE TABLE Joueurs(  
  numjoueur NUMERIC(3,0) CONSTRAINT pk_joueurs PRIMARY KEY,  
  nom VARCHAR(30) NOT NULL,  
  prenom VARCHAR(30),  
  codeequipe CHAR(3),  
  CONSTRAINT fk_Joueurs_equipes FOREIGN KEY (codeequipe) REFERENCES equipes(codeequipe)  
    ON DELETE CASCADE ON UPDATE CASCADE  
);
```

MODIFICATIONS STRUCTURELLES (ALTER TABLE)

➤ Ajout de colonnes

la directive **ADD** de l'instruction ALTER TABLE permet d'ajouter

```
ALTER TABLE Joueurs ADD COLUMN NbButs int DEFAULT 0;
```

➤ Renommer des colonnes

```
ALTER TABLE Joueurs RENAME COLUMN NbButs TO Goals;
```

➤ Modifier le type des colonnes

```
ALTER TABLE Joueurs ALTER COLUMN Goals Numeric(3);
```

➤ Supprimer la colonne

L'option DROP de l'instruction ALTER TABLE permet de supprimer une colonne.

```
ALTER TABLE Joueurs DROP COLUMN Goals
```

ALTER TABLE : SUPPRESSION/AJOUT CONTRAINTE

➤ suppression de contrainte

```
ALTER TABLE Joueurs DROP CONSTRAINT fk_Joueurs_equipes;
```

➤ Ajout de contrainte

- ```
ALTER TABLE Joueurs ADD CONSTRAINT fk_Joueurs_equipes FOREIGN KEY (codeequipe) REFERENCES equipes(codeequipe) ON DELETE CASCADE ON UPDATE CASCADE;
```
- ```
ALTER TABLE Joueurs ADD CONSTRAINT CK_Goals CHECK (Goals >= 0)
```

➤ Renommer une table

```
ALTER TABLE Joueurs RENAME TO Players
```

➤ Supprimer une table

```
DROP TABLE Players
```