TP1 — Tutoriel PostgreSQL

(Pizzerias)

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 (BDDA)

6 Novembre 2025

Version v1.1 (Révision du 07/11/2025) — 2025-11-14 à 11:09:27

Plan

Consignes de travail

Objectif

- Apprendre les bases du langage SQL à partir du modèle `Pizzerias`.
- Comprendre les étapes de création, de manipulation et d'interrogation d'une base relationnelle.

Consigne générale

- Exécuter chaque commande séparément dans `psql`.
- Lire le message affiché par PostgreSQL :
  - Succès → passer à l'étape suivante.
  - Erreur → analyser, corriger, puis réessayer.
- Les lignes commençant par – sont des commentaires explicatifs : ne pas les ignorer.

Ordre logique de travail

- `CREATE ROLE` → `CREATE DATABASE` → `CREATE TABLE` → `INSERT INTO` → `SELECT`
- Conserver une copie du script corrigé pour validation.

Connexion avec psql

Connexion à PostgreSQL sous Windows

```
Win + R -> cmd
Microsoft Windows [Version 10.0.17763.1697]
(C) 2018 Microsoft Corporation. All rights reserved.

C:\Users\danny>chcp 1252
Active code page: 1252

C:\Users\danny>psql -U postgres
Password for user postgres:
```

Configuration côté client (dans psql)

```
psql (16.10)
Type "help" for help.

postgres=# \encoding UTF8
postgres=# SHOW client_encoding;
 client_encoding
-----------------
 UTF8
(1 row)
```

Remarque : UTF8 permet d'afficher correctement les caractères accentués.

Réinitialiser l'état

Nettoyage de l'environnement (dans psql)

```
-- On revient d'abord a la base par defaut "postgres"
postgres=# \c postgres
You are now connected to database "postgres" as user "postgres".

-- Suppression de la base d'exemple s'il en reste une ancienne
postgres=# DROP DATABASE pizzerias;
ERROR:  database "pizzerias" does not exist  -- message normal si deja supprimee

-- Suppression du role utilise pour les tests
postgres=# DROP ROLE laidi;
ERROR:  role "laidi" does not exist          -- message normal si deja supprime
```

Remarque : le prompt `postgres=#` indique que vous etes connecte en tant que superutilisateur.

Rôle, base et droits

```
-- Création d'un rôle utilisateur avec mot de passe (connexion autorisée)
CREATE ROLE laidi LOGIN PASSWORD '1234';

-- Création de la base pizzerias appartenant au rôle "laidi"
CREATE DATABASE pizzerias OWNER laidi;

-- Connexion avec le superutilisateur uniquement pour configurer
postgres=# \c pizzerias
You are now connected to database pizzeriasas user postgres.

-- Attribution des droits sur le schéma "public" :
--   ALL     → lecture et création d'objets
--   USAGE   → utilisation des objets existants
--   CREATE  → création de nouveaux objets (tables, vues, etc.)
GRANT ALL ON SCHEMA public TO laidi;

-- Être propriétaire de la base ne suffit pas pour créer des tables. Par défaut,
-- le schéma "public" appartient à postgres. Il faut donc transférer
-- sa propriété ou le recréer sous le rôle "laidi" :
--       DROP SCHEMA public;
--       CREATE SCHEMA public AUTHORIZATION laidi;
--   Après cette opération, "laidi" peut créer librement tables et vues,
--   sans besoin de GRANT supplémentaire.
ALTER SCHEMA public OWNER TO laidi;
```

Session de travail

Fermer la session actuelle et se reconnecter sous l'utilisateur `laidi`.

1. Quitter la session superutilisateur (dans `psql`)

```
postgres=# \q
```

2. Ouvrir une nouvelle session sous `laidi` (dans `cmd`)

```
C:\> psql -U laidi -d pizzerias
Mot de passe pour l'utilisateur laidi:
psql (16.10)
Saisissez "help" pour l'aide.
```

3. Vérifier la connexion active (dans `psql`)

```
pizzerias=> \conninfo
Vous êtes connecté à la base de données pizzerias en tant qu'utilisateur "laidi"
      sur l'hôte "localhost" (adresse "::1") via le port "5432".
```

Remarque : toutes les commandes suivantes s'exécutent désormais sous `laidi`.

Chaînes de caractères

Définition

Standard SQL :

- `CHAR(n)` — texte à longueur fixe
- `VARCHAR(n)` — texte à longueur variable
- `CLOB` — texte volumineux (Character Large Object)

Utilisation

- `CHAR(n)` pour les codes courts et normalisés (ex. pays, catégories).
- `VARCHAR(n)` pour les libellés et champs textuels classiques.
- `CLOB` pour les contenus longs (commentaires, descriptions détaillées).

Non standard (à éviter) : `TEXT`, `LONGVARCHAR`.

Conseil : fixer des tailles cohérentes pour `VARCHAR(n)` afin d'améliorer tri et indexation.

Types numériques

Définition

Catégories principales :

- **Entiers :** `SMALLINT`, `INTEGER`, `BIGINT`.
- **Décimaux exacts :** `DECIMAL(p,s)`, `NUMERIC(p,s)` — utilisés pour montants ou stocks.
- **Flottants :** `REAL`, `DOUBLE PRECISION` — pour mesures physiques ou calculs scientifiques.

Conseil : privilégier `DECIMAL/NUMERIC` pour la précision des calculs financiers.

Dates et heures

Définition

Types temporels standard :

- `DATE` — jour, mois, année
- `TIME` — heure du jour
- `TIMESTAMP` — date et heure combinées
- `INTERVAL` — durée ou écart temporel

Usage :

- `TIMESTAMP` pour enregistrer les événements.
- `INTERVAL` pour les calculs de durées.

Conseil : préférer `INTERVAL` à des entiers "minutes" ou "secondes".

Auto-incrément (IDENTITY)

Définition

Standard SQL : `GENERATED AS IDENTITY`.

Modes :

- `ALWAYS` — valeur toujours générée automatiquement.
- `BY DEFAULT` — valeur générée sauf si fournie manuellement.

Extension PostgreSQL (non standard) : `SERIAL`.

Exemples de déclaration

```
id INTEGER GENERATED ALWAYS AS IDENTITY PRIMARY KEY
id INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY
```

Conseil : utiliser `IDENTITY` pour la portabilité des scripts SQL.

Contraintes d'intégrité

Rôle des contraintes

En SQL, les **contraintes d'intégrité** assurent la validité, la cohérence et la fiabilité des données. Elles sont vérifiées directement par le moteur du SGBD lors des opérations d'`INSERT`, `UPDATE` ou `DELETE`, garantissant que les règles définies dans le schéma ne sont jamais violées.

Types de contraintes (standard SQL)

- `NOT NULL` — interdit les valeurs absentes.
- `UNIQUE` — empêche les doublons.
- `PRIMARY KEY` — identifie chaque ligne de manière unique.
- `FOREIGN KEY` — crée un lien de dépendance entre deux tables.
- `CHECK` — impose une condition logique sur les valeurs.
- `CONSTRAINT` — permet de nommer explicitement une contrainte.

Remarque : les contraintes font partie intégrante du modèle relationnel.

Contraintes d'intégrité(suite)

Exemples de contraintes

```
CONSTRAINT uq_email UNIQUE(email)
FOREIGN KEY(id_client) REFERENCES client(id)
CONSTRAINT ck_prix_positif CHECK(prix >= 0)
```

- Nommer clairement chaque contrainte pour faciliter la maintenance et la lecture du schéma.
- Définir les comportements `ON DELETE` et `ON UPDATE` selon la logique métier.
- Préférer les contraintes SQL aux vérifications dans le code applicatif.
- Placer les contraintes à la fin des instructions `CREATE TABLE` pour plus de lisibilité.

Conseil : préciser les actions de clé étrangère — par ex. `ON DELETE CASCADE` ou `SET NULL`.

Création de la table personne

Version 1 – Écriture directe (compacte)

```
CREATE TABLE personne (
  nom   VARCHAR(20) PRIMARY KEY,
  age   SMALLINT CHECK (age BETWEEN 0 AND 120),
  genre CHAR(1) CHECK (genre IN ('H','F'))
);
```

Version 2 – Écriture structurée (préférée)

```
CREATE TABLE personne (
  nom    VARCHAR(20),
  age    SMALLINT,
  genre  CHAR(1),

  -- Contraintes structurelles regroupées à la fin
  CONSTRAINT personne_pkey PRIMARY KEY (nom),
  CONSTRAINT personne_age_ck   CHECK (age BETWEEN 0 AND 120),
  CONSTRAINT personne_genre_ck CHECK (genre IN ('H','F'))
);
```

Création de la table `frequente`

Version 1 – Écriture directe (compacte)

```
CREATE TABLE frequente (
  nom       VARCHAR(20) NOT NULL,
  pizzeria  VARCHAR(30) NOT NULL,
  PRIMARY KEY (nom, pizzeria),
  FOREIGN KEY (nom) REFERENCES personne(nom)
);
```

Version 2 – Écriture structurée (préférée)

```
CREATE TABLE frequente (
  nom       VARCHAR(20) NOT NULL,
  pizzeria  VARCHAR(30) NOT NULL,

  -- Contraintes structurelles regroupées à la fin
  CONSTRAINT frequente_pkey PRIMARY KEY (nom, pizzeria),
  CONSTRAINT frequente_nom_fkey FOREIGN KEY (nom)
    REFERENCES personne(nom)
);
```

Création de la table mange

Version 1 – Écriture directe (compacte)

```
CREATE TABLE mange (
  nom   VARCHAR(20) NOT NULL,
  pizza VARCHAR(60) NOT NULL,
  PRIMARY KEY (nom, pizza),
  FOREIGN KEY (nom) REFERENCES personne(nom)
);
```

Version 2 – Écriture structurée (préférée)

```
CREATE TABLE mange (
  nom    VARCHAR(20) NOT NULL,
  pizza  VARCHAR(60) NOT NULL,

  -- Contraintes structurelles regroupées à la fin
  CONSTRAINT mange_pkey PRIMARY KEY (nom, pizza),
  CONSTRAINT mange_nom_fkey FOREIGN KEY (nom)
    REFERENCES personne(nom)
);
```

Création de la table sert

Version 1 – Écriture directe (compacte)

```
CREATE TABLE sert (
  pizzeria VARCHAR(30) NOT NULL,
  pizza    VARCHAR(60) NOT NULL,
  prix     NUMERIC(10,2) CHECK (prix >= 0),
  PRIMARY KEY (pizzeria, pizza)
);
```

Version 2 – Écriture structurée (préférée)

```
CREATE TABLE sert (
  pizzeria  VARCHAR(30) NOT NULL,
  pizza     VARCHAR(60) NOT NULL,
  prix      NUMERIC(10,2),

  -- Contraintes structurelles regroupées à la fin
  CONSTRAINT sert_pkey PRIMARY KEY (pizzeria, pizza),
  CONSTRAINT sert_prix_ck CHECK (prix >= 0)
);
```

Mineurs et pizzerias fréquentées

Énoncé : Renvoyer la liste des pizzerias fréquentées par au moins une personne de moins de 18 ans.

```sql
-- On relie `frequente` (nom <-> pizzeria) et `personne` (nom <-> âge)
-- puis on filtre sur l'âge < 18. DISTINCT évite les doublons de pizzerias.

SELECT DISTINCT f.pizzeria
FROM frequente f
JOIN personne p ON p.nom = f.nom
WHERE p.age < 18;
```

Femmes mangeant champignon ou thon

Énoncé : Noms distincts des personnes de genre 'F' qui mangent 'champignon' ou 'thon'.

```sql
-- Filtre de genre côté `personne`, puis test d'appartenance IN côté `mange`.

SELECT DISTINCT p.nom
FROM personne p
JOIN mange m ON m.nom = p.nom
WHERE p.genre = 'F'
  AND m.pizza IN ('champignon','thon');
```

Femmes mangeant les deux pizzas

Énoncé : Noms des femmes qui mangent à la fois 'champignon' et 'thon'.

```sql
-- On regroupe par personne et on exige deux pizzas distinctes dans l'ensemble
     cible.

SELECT p.nom
FROM personne p
JOIN mange m ON m.nom = p.nom
WHERE p.genre = 'F'
  AND m.pizza IN ('champignon','thon')
GROUP BY p.nom
HAVING COUNT(DISTINCT m.pizza) = 2;
```

Pizzerias d'Amy à $<$ 1000 DA

Énoncé : Pizzerias servant au moins une pizza qu'Amy mange à un prix strictement $<$ 1000 DA.

```sql
-- `mange` relie Amy à ses pizzas ; `sert` relie pizzerias et pizzas avec les
     prix.

SELECT DISTINCT s.pizzeria
FROM sert s
JOIN mange m ON m.pizza = s.pizza
WHERE m.nom = 'Amy'
  AND s.prix < 1000;
```

Pizzerias mono-genre

Énoncé : Pizzerias dont les clients fréquentants sont tous du même genre (H ou F).

```sql
-- Une pizzeria est mono-genre si COUNT(DISTINCT p.genre) = 1 dans son groupe.

SELECT f.pizzeria
FROM frequente f
JOIN personne p ON p.nom = f.nom
GROUP BY f.pizzeria
HAVING COUNT(DISTINCT p.genre) = 1;
```

Pizzas non servies dans leurs pizzerias

Énoncé : (nom, pizza) telles qu'aucune pizzeria fréquentée par la personne ne sert cette pizza.

```sql
-- Schéma NOT EXISTS : on conserve la paire si aucune pizzeria fréquentée ne la
     propose.

SELECT m.nom, m.pizza
FROM mange m
WHERE NOT EXISTS (
  SELECT 1
  FROM frequente f
  JOIN sert s ON s.pizzeria = f.pizzeria
             AND s.pizza    = m.pizza
  WHERE f.nom = m.nom
);
```

Pizzerias cohérentes

Énoncé : Personnes ne fréquentant que des pizzerias servant au moins une de leurs pizzas.

```sql
-- Pour toute pizzeria fréquentée, il doit exister une pizza de la personne
     servie dans cette pizzeria.

SELECT p.nom
FROM personne p
WHERE NOT EXISTS (
  SELECT 1
  FROM frequente f
  WHERE f.nom = p.nom
    AND NOT EXISTS (
      SELECT 1
      FROM sert s
      JOIN mange m ON m.nom = p.nom AND m.pizza = s.pizza
      WHERE s.pizzeria = f.pizzeria
    )
);
```

Fréquentent toutes les pizzerias pertinentes

Énoncé : Personnes qui fréquentent toutes les pizzerias servant au moins une pizza qu'elles mangent.

```sql
-- "Pour toute pizzeria qui sert une pizza de la personne, la personne la fré
     quente" (double NOT EXISTS).

SELECT p.nom
FROM personne p
WHERE NOT EXISTS (
  SELECT 1
  FROM sert s
  JOIN mange m ON m.nom = p.nom AND m.pizza = s.pizza
  WHERE NOT EXISTS (
    SELECT 1
    FROM frequente f
    WHERE f.nom = p.nom
      AND f.pizzeria = s.pizzeria
  )
);
```

Prix minimal pour 'thon'

Énoncé : Pizzerias proposant la pizza 'thon' au prix minimal observé.

```sql
-- On compare le prix de 'thon' au minimum global sur cette pizza.

SELECT s.pizzeria
FROM sert s
WHERE s.pizza = 'thon'
  AND s.prix  = (
    SELECT MIN(prix)
    FROM sert
    WHERE pizza = 'thon'
  );
```