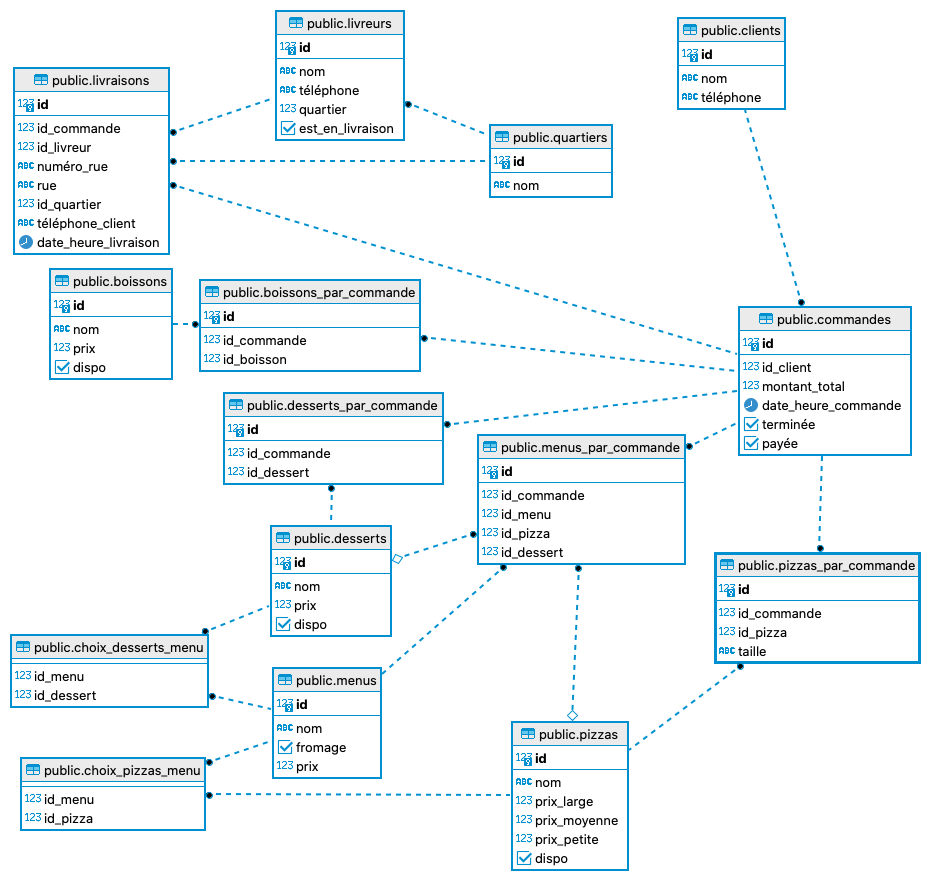
TP NFP 107

Par Jérémy et Marc

<https://github.com/d4wae89d498/NFP107-TP>

**Text

Description automatically generated**

Sommaire

[Technologies 3](#_Toc123996848)

[Automatisations 3](#_Toc123996849)

[Création des tables 4](#_Toc123996850)

[Jeu de données : les quartiers lyonnais et des menus variés 7](#_Toc123996851)

[la fonction commander 9](#_Toc123996852)

[Autres requêtes 13](#_Toc123996853)

[Notes 16](#_Toc123996854)

# Technologies

Nous avons choisi PostgreSQL car c’est système de gestion de base de données de haute performance disponible en source ouverte que nous souhaitons maitriser davantage. Le diagramme a été créé à l’aide de DBeaver. Nous avons aussi utilisé PgAdmin4 pour visualiser nos requêtes.

# Automatisations

Nous avons automatisé une partie du processus de livraison avec une fonction « commander »  qui prends en paramètre le nom du client, ce qui est commandé en menu ou en extra, et éventuellement des coordonnées de livraison. Notre fonction retourne un identifiant de commande et l’identifiant du livreur.

Si la commande est à livrer : notre fonction vérifie qu’un livreur est disponible pour le quartier demandé, si oui un livreur est automatiquement assigné, sinon une exception est levée.

Notre fonction « commander » vérifie également la cohérence des données afin de donner des messages d’erreurs clairs. De plus, dans le cadre des pizzas hors-menu ; le client peut choisir plusieurs tailles (petite, moyenne, large). Les pizzas en menus sont toutes des pizzas larges.

# Création des tables

-- Créer les tables

CREATE TABLE pizzas (

id SERIAL PRIMARY KEY,

nom VARCHAR(255) NOT NULL UNIQUE,

prix\_large NUMERIC(5,2) NOT NULL,

prix\_moyenne NUMERIC(5,2) NOT NULL,

prix\_petite NUMERIC(5,2) NOT NULL,

dispo BOOLEAN NOT NULL DEFAULT(true)

);

CREATE TABLE menus (

id SERIAL PRIMARY KEY,

nom VARCHAR(255) NOT NULL UNIQUE,

fromage BOOLEAN NOT NULL,

prix DECIMAL(5,2) NOT NULL

);

CREATE TABLE choix\_pizzas\_menu (

id\_menu INTEGER NOT NULL REFERENCES menus (id),

id\_pizza INTEGER NOT NULL REFERENCES pizzas (id)

);

CREATE TABLE desserts (

id SERIAL PRIMARY KEY,

nom VARCHAR(255) NOT NULL UNIQUE,

prix DECIMAL(5,2) NOT NULL,

dispo BOOLEAN NOT NULL DEFAULT(true)

);

CREATE TABLE choix\_desserts\_menu (

id\_menu INTEGER NOT NULL REFERENCES menus (id),

id\_dessert INTEGER NOT NULL REFERENCES desserts (id)

);

CREATE TABLE clients (

id SERIAL PRIMARY KEY,

nom VARCHAR(255) NOT NULL UNIQUE,

téléphone VARCHAR(15) NULL

);

CREATE TABLE commandes (

id SERIAL PRIMARY KEY,

id\_client INTEGER NOT NULL REFERENCES clients (id),

montant\_total DECIMAL(10,2) NOT NULL,

date\_heure\_commande TIMESTAMP NOT NULL default(NOW()),

terminée BOOLEAN NOT NULL DEFAULT(false),

payée BOOLEAN NOT NULL DEFAULT(false)

);

CREATE TABLE menus\_par\_commande (

id SERIAL PRIMARY KEY,

id\_commande INTEGER NOT NULL REFERENCES commandes (id),

id\_menu INTEGER NOT NULL REFERENCES menus (id),

id\_pizza INTEGER NULL REFERENCES pizzas (id),

id\_dessert INTEGER NULL REFERENCES desserts (id)

);

CREATE TABLE pizzas\_par\_commande (

id SERIAL PRIMARY KEY,

id\_commande INTEGER NOT NULL REFERENCES commandes (id),

id\_pizza INTEGER NOT NULL REFERENCES pizzas (id),

taille VARCHAR(255) NOT NULL

);

CREATE TABLE desserts\_par\_commande (

id SERIAL PRIMARY KEY,

id\_commande INTEGER NOT NULL REFERENCES commandes (id),

id\_dessert INTEGER NOT NULL REFERENCES desserts (id)

);

CREATE TABLE boissons (

id SERIAL PRIMARY KEY,

nom VARCHAR(255) NOT NULL UNIQUE,

prix DECIMAL(5,2) NOT NULL,

dispo BOOLEAN NOT NULL DEFAULT(true)

);

CREATE TABLE boissons\_par\_commande (

id SERIAL PRIMARY KEY,

id\_commande INTEGER NOT NULL REFERENCES commandes (id),

id\_boisson INTEGER NOT NULL REFERENCES boissons (id)

);

CREATE TABLE quartiers (

id SERIAL PRIMARY KEY,

nom VARCHAR (255) NOT NULL UNIQUE

);

CREATE TABLE livreurs (

id SERIAL PRIMARY KEY,

nom VARCHAR (255) NOT NULL UNIQUE,

téléphone VARCHAR (255) NOT NULL,

quartier INTEGER NOT NULL REFERENCES quartiers (id),

est\_en\_livraison BOOLEAN NOT NULL default (false)

);

CREATE TABLE livraisons (

id SERIAL PRIMARY KEY,

id\_commande INTEGER NOT NULL REFERENCES commandes (id),

id\_livreur INTEGER NOT NULL REFERENCES livreurs (id),

numéro\_rue VARCHAR(255) NOT NULL,

rue VARCHAR(255) NOT NULL,

id\_quartier INTEGER NOT NULL REFERENCES quartiers (id),

téléphone\_client VARCHAR (255) NOT NULL,

date\_heure\_livraison TIMESTAMP NULL

);

# Jeu de données : les quartiers lyonnais et des menus variés

-- Transaction pour insérer les données

BEGIN TRANSACTION;

INSERT INTO pizzas (nom, prix\_large, prix\_moyenne, prix\_petite, dispo)

VALUES

('4 fromages', 12.50, 9.50, 7.50, true),

('Margherita', 10.50, 8.50, 6.50, true),

('Pepperoni', 11.50, 9.50, 7.50, true),

('Végétarienne', 11.00, 8.50, 6.50, true),

('BBQ New York', 12.50, 9.50, 7.50, true),

('Merguez', 13.50, 10.50, 8.50, true),

('Raclette', 12.00, 9.00, 7.00, true);

INSERT INTO boissons (nom, prix, dispo)

VALUES

('Coca', 2.00, true),

('1664', 3.00, true),

('Sprite', 2.00, true),

('Ice Tea', 2.00, true);

INSERT INTO desserts (nom, prix, dispo)

VALUES

('Tiramisu', 3.50, true),

('Panna cotta', 3.50, true),

('Mousse au chocolat', 3.50, true),

('Glace', 3.50, true),

('Macarons crème', 4.00, true),

('Banane au whisky', 12.00, true);

INSERT INTO menus (nom, fromage, prix)

VALUES

('Printemps', true, 12.50),

('Eté', false, 11.50),

('Automne', true, 11.00),

('Hiver', true, 12.00);

INSERT INTO choix\_pizzas\_menu (id\_menu, id\_pizza)

VALUES

((SELECT id FROM menus WHERE nom = 'Printemps'),

(SELECT id FROM pizzas WHERE nom = 'Margherita')),

((SELECT id FROM menus WHERE nom = 'Eté'),

(SELECT id FROM pizzas WHERE nom = 'Merguez')),

((SELECT id FROM menus WHERE nom = 'Automne'),

(SELECT id FROM pizzas WHERE nom = 'BBQ New York')),

((SELECT id FROM menus WHERE nom = 'Hiver'),

(SELECT id FROM pizzas WHERE nom = '4 fromages')),

((SELECT id FROM menus WHERE nom = 'Hiver'),

(SELECT id FROM pizzas WHERE nom = 'Raclette'));

INSERT INTO choix\_desserts\_menu (id\_menu, id\_dessert)

VALUES

((SELECT id FROM menus WHERE nom = 'Printemps'),

(SELECT id FROM desserts WHERE nom = 'Tiramisu')),

((SELECT id FROM menus WHERE nom = 'Eté'),

(SELECT id FROM desserts WHERE nom = 'Glace')),

((SELECT id FROM menus WHERE nom = 'Automne'),

(SELECT id FROM desserts WHERE nom = 'Panna cotta')),

((SELECT id FROM menus WHERE nom = 'Hiver'),

(SELECT id FROM desserts WHERE nom = 'Mousse au chocolat')),

((SELECT id FROM menus WHERE nom = 'Hiver'),

(SELECT id FROM desserts WHERE nom = 'Banane au whisky'));

insert into quartiers (id, nom)

VALUES (1, 'Hotel de Ville'),

(2, 'Bellecours-Ainay-Confluence'),

(3, 'Part-Dieu'),

(4, 'Croix Rousse'),

(5, 'Vieux Lyon'),

(6, 'Foche'),

(7, 'Mermoz'),

(8, 'Etats-Unis');

insert into livreurs (nom, téléphone, quartier, est\_en\_livraison)

VALUES

('Jean Paul', '0644889865', 1, false),

('Charles Henry', '0647942304', 2, false),

('Pierre', '0642989584', 3, false),

('Alice', '0652495849', 4, false),

('Bob', '0640408952', 5, false),

('Rose', '0740239944', 6, false),

('Agathe', '0642342442', 7, false),

('Christophe', '0642898753', 8, false);

COMMIT;

# la fonction commander

CREATE OR REPLACE FUNCTION public.commander(

nom\_client character varying,

liste\_menus character varying[],

liste\_desserts\_menus character varying[],

liste\_pizzas\_menus character varying[],

liste\_pizzas character varying[],

liste\_tailles\_pizzas character varying[],

liste\_desserts character varying[],

liste\_boissons character varying[],

"à\_emporter" boolean,

"téléphone\_client" character varying,

"numéro\_rue" character varying,

rue character varying,

quartier character varying,

OUT id\_commande integer,

OUT id\_livreur integer)

RETURNS record

LANGUAGE 'plpgsql'

COST 100

VOLATILE PARALLEL UNSAFE

AS $BODY$

DECLARE id\_client INTEGER;

DECLARE id\_du\_menu INTEGER;

DECLARE id\_du\_quartier INTEGER;

BEGIN

id\_commande := null;

id\_livreur := null;

-- Enregistrer un client

if (nom\_client is null) then

raise exception 'Le nom du client est obligatoire';

end if;

if not exists (select 1 from clients where nom = nom\_client) then

insert into clients (nom, téléphone) values (nom\_client, téléphone\_client);

else

update clients set téléphone = téléphone\_client where nom = nom\_client;

end if;

id\_client := (select id from clients where nom = nom\_client);

-- Enregistrer la commande et verifier la cohérance

if liste\_menus is null and liste\_pizzas is null and liste\_desserts is null and liste\_boissons is null then

raise exception 'Il n''y a rien a commander!';

end if;

if

liste\_menus is not null and liste\_desserts\_menus is not null and cardinality(liste\_menus) != cardinality(liste\_desserts\_menus)

or

liste\_menus is not null and liste\_desserts\_menus is null

or

liste\_menus is null and liste\_desserts\_menus is not null

or

liste\_menus is not null and liste\_pizzas\_menus is not null and cardinality(liste\_menus) != cardinality(liste\_pizzas\_menus)

or

liste\_menus is not null and liste\_pizzas\_menus is null

or

liste\_menus is null and liste\_pizzas\_menus is not null

then

raise exception 'Le nombre de menu et de desserts/pizzas par menu doit etre identique!';

end if;

insert into commandes (id\_client, montant\_total, terminée, payée)

values (id\_client, 0, false, false) returning id into id\_commande;

if liste\_menus is not null then

for i in 1..cardinality(liste\_menus) loop

if not exists (select \* from menus where nom = liste\_menus[i]) then

raise exception 'Le menu n''existe pas!';

end if;

id\_du\_menu = (select id from menus where nom = liste\_menus[i]);

insert into menus\_par\_commande (id\_commande, id\_menu)

values (id\_commande, id\_du\_menu);

if liste\_pizzas\_menus[i] is null then

raise exception 'Un menu doit avoir une pizza!';

end if;

if not exists (select \* from choix\_pizzas\_menu

where id\_menu = id\_du\_menu and id\_pizza = (

select id from pizzas where nom = liste\_pizzas\_menus[i])) then

raise exception 'La pizza n''existe pas dans le menu!';

end if;

update menus\_par\_commande set id\_pizza = (

select id from pizzas where nom = liste\_pizzas\_menus[i]

) where id\_menu = id\_du\_menu;

if liste\_desserts\_menus[i] is not null then

if not exists (select \* from choix\_desserts\_menu

where id\_menu = id\_du\_menu and id\_dessert = (

select id from desserts where nom = liste\_desserts\_menus[i])) then

raise exception 'Le déssert n''existe pas dans le menu!';

end if ;

update menus\_par\_commande set id\_dessert = (

select id from desserts where nom = liste\_desserts\_menus[i]

) where id\_menu = id\_du\_menu;

end if;

update commandes

set montant\_total = montant\_total + (SELECT prix FROM menus WHERE nom = liste\_menus[i])

where id = id\_commande;

end loop;

end if;

-- Extras hors menu

if liste\_pizzas is not null then

for i in 1..cardinality(liste\_pizzas) loop

if not exists (select \* from pizzas where nom = liste\_pizzas[i]) then

raise exception 'Il y a une pizza dans lise\_pizza qui n''existe pas';

end if;

insert into pizzas\_par\_commande (id\_commande, id\_pizza, taille) values (id\_commande, (select id from pizzas where nom = liste\_pizzas[i]), liste\_tailles\_pizzas[i]);

if (liste\_tailles\_pizzas[i] = 'petite') then

update commandes

set montant\_total = montant\_total + (SELECT prix\_petite FROM pizzas WHERE nom = liste\_pizzas[i])

where id = id\_commande;

elsif (liste\_tailles\_pizzas[i] = 'moyenne') then

update commandes

set montant\_total = montant\_total + (SELECT prix\_moyenne FROM pizzas WHERE nom = liste\_pizzas[i])

where id = id\_commande;

elsif (liste\_tailles\_pizzas[i] = 'grande') then

update commandes

set montant\_total = montant\_total + (SELECT prix\_grande FROM pizzas WHERE nom = liste\_pizzas[i])

where id = id\_commande;

else

raise exception 'Il manque des taille pour les pizzas hors menu!';

end if;

end loop;

end if;

if liste\_desserts is not null then

for i in 1..cardinality(liste\_desserts) loop

if not exists (select \* from desserts where nom = liste\_desserts[i]) then

raise exception 'Il y a un dessert dans liste\_desserts qui n''existe pas';

end if;

update commandes

set montant\_total = montant\_total + (SELECT prix FROM desserts WHERE nom = liste\_desserts[i])

where id = id\_commande;

insert into desserts\_par\_commande (id\_commande, id\_dessert) values (id\_commande, (select id from desserts where nom = liste\_desserts[i]));

end loop;

end if;

if liste\_boissons is not null then

for i in 1..cardinality(liste\_boissons) loop

if not exists (select \* from boissons where nom = liste\_boissons[i]) then

raise exception 'Il y a une boisson dans liste\_boissons qui n''existe pas';

end if;

update commandes

set montant\_total = montant\_total + (SELECT prix FROM boissons WHERE nom = liste\_boissons[i])

where id = id\_commande;

insert into boissons\_par\_commande (id\_commande, id\_boisson) values (id\_commande, (select id from boissons where nom = liste\_boissons[i]));

end loop;

end if;

-- Gerer la partie livraison

if à\_emporter then

if téléphone\_client is null or numéro\_rue is null or rue is null is null or quartier is null then

raise exception 'Les données de livraison sont incomplètes';

end if;

id\_du\_quartier := (select id from quartiers where nom = quartier);

id\_livreur := (select id from livreurs where livreurs.quartier = id\_du\_quartier limit 1);

if id\_livreur is null then

raise exception 'Aucun livreur disponible dans le quartier demandé';

end if;

insert into livraisons (id\_commande, id\_livreur, numéro\_rue, rue, id\_quartier, téléphone\_client)

values (id\_commande, id\_livreur, numéro\_rue, rue, id\_du\_quartier, téléphone\_client);

end if;

END;

$BODY$;

# Autres requêtes

-- passer une commande

SELECT \* from public.commander(

nom\_client => 'John Doe',

-- LES MENUS ET LES OPTIONS DES MENUS

liste\_menus => array['Hiver', 'Eté'],

liste\_desserts\_menus => array['Banane au whisky', 'Glace'],

liste\_pizzas\_menus => array ['4 fromages', 'Merguez'],

-- LES EXTRAS

liste\_pizzas => array['Raclette'],

liste\_tailles\_pizzas => array['petite'],

liste\_desserts => array['Tiramisu'],

liste\_boissons => array['1664', '1664', '1664', 'Sprite'],

-- DETAILS COMMANDE

à\_emporter => true,

téléphone\_client => '07 88 09 12 35',

numéro\_rue => '35B',

rue => 'Rue Paul Santy',

quartier => 'Mermoz'

)



-- liste des menus commandés

select menus.nom as nom\_menu, pizzas.nom as nom\_pizza,

desserts.nom as nom\_dessert, fromage, menus.prix

from menus\_par\_commande, menus, pizzas, desserts

where

id\_commande = 4

and id\_menu = menus.id

and pizzas.id = id\_pizza

and desserts.id = id\_dessert

Table

Description automatically generated

-- liste pizzas en extra

select pizzas.nom, taille from pizzas\_par\_commande, pizzas

where

pizzas.id = id\_pizza

and id\_commande = 4

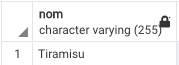
-- liste desserts en extra

select desserts.nom from desserts\_par\_commande, desserts

where

desserts.id = id\_dessert

and id\_commande = 4



-- liste boissons en extra

select boissons.nom from boissons\_par\_commande, boissons

where

boissons.id = id\_boisson

and id\_commande = 4

Table

Description automatically generated

-- le total de la commande

select montant\_total from commandes where id = 4



-- Un livreur est arrivée chez un client

BEGIN TRANSACTION;

update public.commande set terminée = true, payée = true where id = 8;

update public.livraison set date\_heure\_livraison = now() where id\_commande = 8;

COMMIT;

-- Example de requête pour le retour d'un livreur à la pizzeria après une livraison

BEGIN TRANSACTION;

update public.livreurs set est\_en\_livraison = false

where nom = 'Rose'

COMMIT;

-- liste des boisons par nb vente decroissant

select boissons.nom, count(boissons\_par\_commande.id) as nb

from public.boissons, boissons\_par\_commande

where boissons.id = id\_boisson

GROUP BY boissons.nom

ORDER BY nb desc



# Notes

Nous avons choisi comme dernière requête les boissons les plus vendus car une requête pour les pizzas les plus vendus, en sachant que les pizzas peuvent être en menu ou en extra aurait été un peu plus compliqué sans plpgsql ou autre.

Nous avons apprécié le langage plpgsql pour notre fonction « commander » uniquement. PostgreSQL est en effet très puissant mais les messages d’erreurs de syntaxe de plpgsql n’ont pas été facile à comprendre pour nous débutants.

Aussi nous avons choisis une fonction et non pas une procédure stockée car nous voulions pouvoir immédiatement utiliser le résultat de notre fonction commander :

select montant\_total from commandes where id = (SELECT id\_commande from

public.commander(

nom\_client => ...

))

Mais seules les procédures stockées peuvent COMMIT leur transaction avant de retourner ; ce n’est pas possible avec une fonction sans utiliser des techniques avancées que nous n’avons pas encore trouvé. Nous avons remarqué qu’il est possible d’écrire des fonctions pour PostgreSQL en langage C directement ; peut-être que cela pourrait nous aider à contourner ce genre de problème.

Graphical user interface, text, application, email

Description automatically generated

Mais … cela dépasserait le cadre ce TP.