Théotime Archas

Hugo Antaki

Sina Balaeigroodi

**PSI :**

**RAPPORT RENDU 1**

Sommaire :

1. Introduction
2. Présentation de notre diagramme entité-association
3. Partie graphe

Lien GitHub : <https://github.com/PTerribleRedactor/Etape_1_projet_PSI.git>

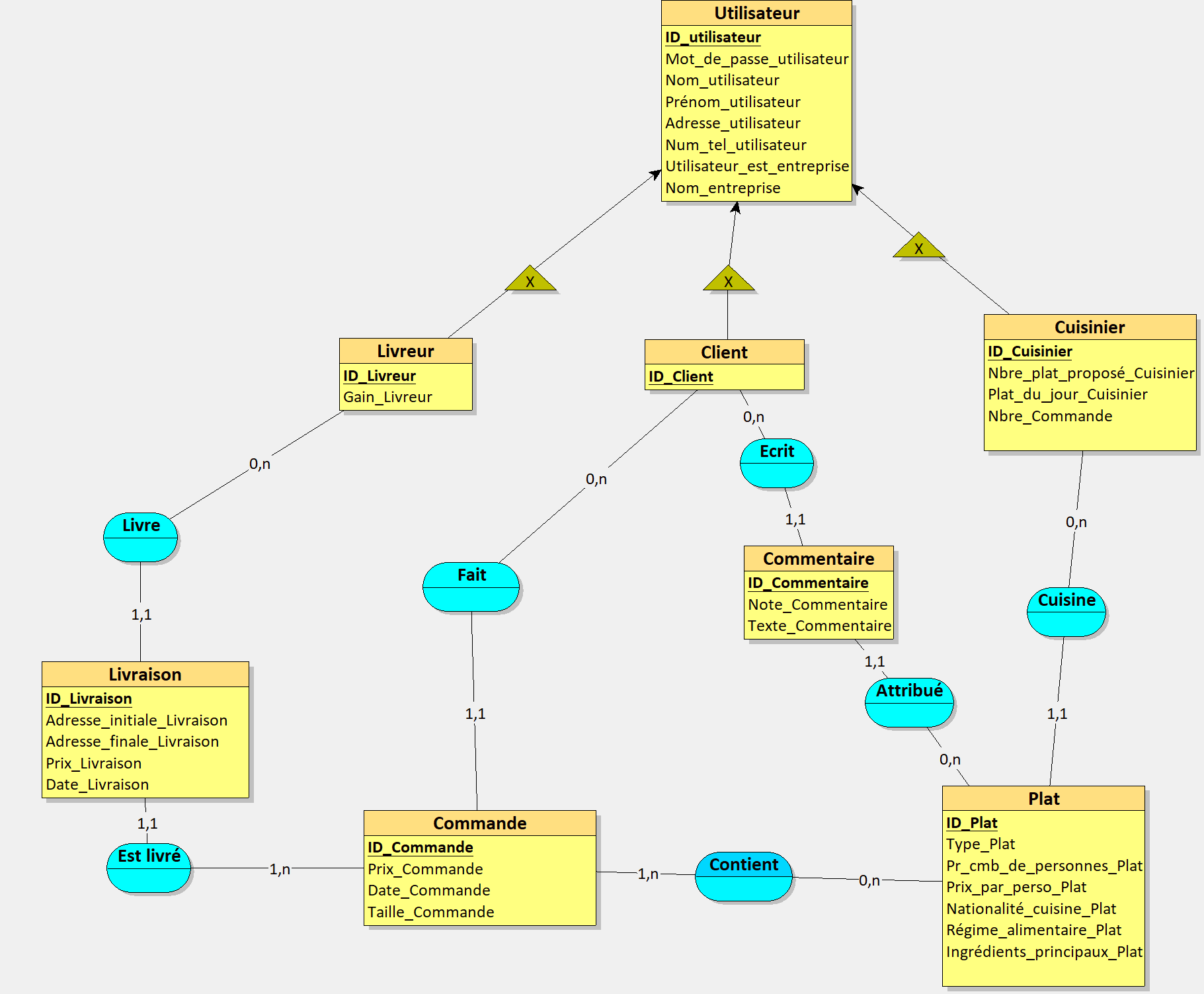
1. Introduction

Le projet Liv'in Paris consiste en la création d'une application permettant le partage de repas entre voisins situés dans Paris intramuros. L'objectif est de connecter des cuisiniers et des clients (particuliers ou entreprises locales) via une plateforme. Les utilisateurs doivent s'inscrire en fournissant des informations personnelles telles que nom, prénom, adresse, numéro de téléphone et email. Les cuisiniers peuvent aussi être clients et vice versa, et l'application génère un identifiant unique pour chaque utilisateur.

Dans ce rendu nous allons présenter la solution de notre modèle entité-association, ainsi que de notre schéma relationnel. Nous allons aussi montrer leur transcription dans MySQL, de même que le peuplement de ces tables et enfin quelques requêtes SELECT.

D’un autre côté, nous exposerons concernant les graphes plusieurs points : Instanciation d'une structure de graphe (Classe Noeud, Lien, Graphe) à partir des adhérents d'un club associatif, Parcours en profondeur et largeur, détection automatique d'un graphe connexe et d'un circuit, visualisation du graphe.

1. Présentation de notre modèle Entité-Association, Schéma relationnel et transcription dans MySQL



Explication du Schéma Looping

Nous avons conçu un modèle Entité-Association pour répondre au mieux au cahier des charges qui nous avait été fourni.

Relations entre les entités :

Nous avons créé plusieurs tables (les propriétés de ces tables sont ci-dessous après les relations), notamment celle Utilisateur, qui contient toutes les informations personnelles demandées (identifiant, nom, prénom, adresse, numéro de téléphone, ...). Cette table regroupe le profil de Particulier et celui d’entreprise avec des variables laissées vides (on configure une entreprise avec les attributs de son représentant + le nom de l’entreprise).

Un utilisateur peut devenir et/ou livreur, cuisinier, client.

Un client passe des Commandes en fonction des plats disponibles.

Un cuisinier prépare des Plats.

Une commande contient plusieurs Plats.

Un plat peut être dans plusieurs commandes (on ne peut pas commander qu’un seule fois de la pizza).

Une livraison est associée à une Commande.

Un commentaire est attribué à un Plat.

(Nous n’avons pas choisi de faire dépendre les commentaires des commandes mais des plats, comme sur les grands sites marchands, où on a la possibilité de laisser des commentaires librement sans devoir acheter le produit).

Ainsi nos tables contiennent les propriétés suivantes que nous avons annotés :

1. Utilisateur :

o ID\_utilisateur : Identifiant unique pour chaque utilisateur.

o Mot\_de\_passe\_utilisateur : Mot de passe pour l'authentification.

o Nom\_utilisateur : Nom de l'utilisateur.

o Prénom\_utilisateur : Prénom de l'utilisateur.

o Adresse\_utilisateur : Adresse de l'utilisateur.

o Num\_tel\_utilisateur : Numéro de téléphone de l'utilisateur.

o Utilisateur\_est\_entreprise : Indicateur si l'utilisateur est une entreprise.

o Nom\_entreprise : Nom de l'entreprise (si applicable).

1. Client :

o ID\_Client : Identifiant unique pour chaque client.

o O.n : Relation avec d'autres entités (non spécifiée dans le schéma).

1. Cuisinier :

o ID\_Cuisinier : Identifiant unique pour chaque cuisinier.

o Nbre\_plat\_propose\_Cuisinier : Nombre de plats proposés par le cuisinier.

o Plat\_du\_jour\_Cuisinier : Plat du jour proposé par le cuisinier.

o Nbre\_Commande : Nombre de commandes réalisées par le cuisinier.

1. Plat :

o ID\_Plat : Identifiant unique pour chaque plat.

o Type\_Plat : Type de plat (entrée, plat principal, dessert).

o Pr\_cmb\_de\_personnes\_Plat : Nombre de personnes pour lesquelles le plat est prévu.

o Prix\_par\_perso\_Plat : Prix par personne.

o Nationalité\_cuisine\_Plat : Nationalité de la cuisine (chinoise, mexicaine, etc.). o Régime\_alimentaire\_Plat : Régime alimentaire (végétarien, sans gluten, etc.).

o Ingrédients\_principaux\_Plat : Ingrédients principaux du plat.

1. Commande :

o ID\_Commande : Identifiant unique pour chaque commande.

o Prix\_Commande : Prix total de la commande.

o Date\_Commande : Date de la commande.

o Taille\_Commande : Taille de la commande (nombre de plats).

1. Livraison :

o ID\_Livraison : Identifiant unique pour chaque livraison.

o Adresse\_initiale\_Livraison : Adresse de départ de la livraison.

o Adresse\_finale\_Livraison : Adresse de destination de la livraison.

o Prix\_Livraison : Prix de la livraison.

o Date\_Livraison : Date de la livraison.

1. Commentaire :

o ID\_Commentaire : Identifiant unique pour chaque commentaire.

o Note\_Commentaire : Note attribuée au commentaire.

o Texte\_Commentaire : Texte du commentaire.

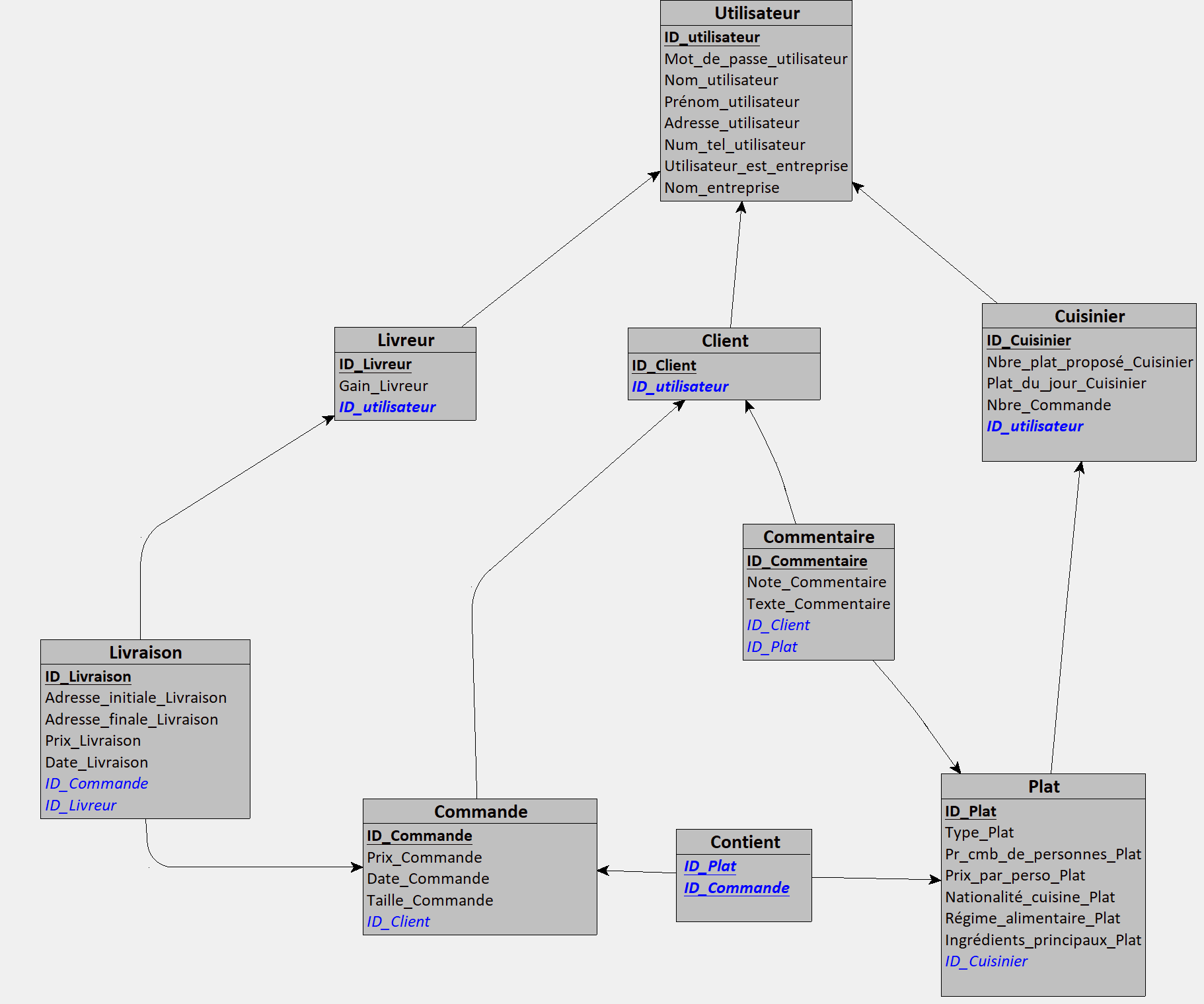
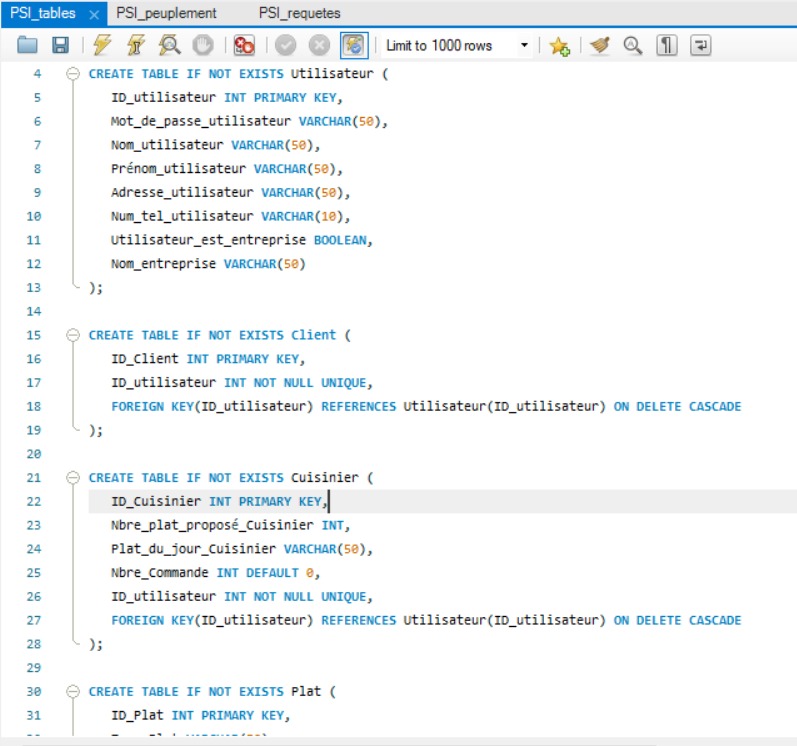


Schéma relationnel :

Utilisateur = (ID\_utilisateur INT, Mot\_de\_passe\_utilisateur VARCHAR(50), Nom\_utilisateur VARCHAR(50), Prénom\_utilisateur VARCHAR(50), Adresse\_utilisateur VARCHAR(50), Num\_tel\_utilisateur VARCHAR(10), Utilisateur\_est\_entreprise LOGICAL, Nom\_entreprise VARCHAR(50)); Client = (ID\_Client INT, #ID\_utilisateur); Cuisinier = (ID\_Cuisinier INT, Nbre\_plat\_proposé\_Cuisinier INT, Plat\_du\_jour\_Cuisinier VARCHAR(50), Nbre\_Commande COUNTER, #ID\_utilisateur); Plat = (ID\_Plat INT, Type\_Plat VARCHAR(50), Pr\_cmb\_de\_personnes\_Plat INT, Prix\_par\_perso\_Plat CURRENCY, Nationalité\_cuisine\_Plat VARCHAR(50), Régime\_alimentaire\_Plat TEXT, Ingrédients\_principaux\_Plat TEXT, #ID\_Cuisinier); Livreur = (ID\_Livreur INT, Gain\_Livreur DECIMAL(15,2), #ID\_utilisateur); Commentaire = (ID\_Commentaire INT, Note\_Commentaire INT, Texte\_Commentaire TEXT, #ID\_Client, #ID\_Plat); Commande = (ID\_Commande INT, Prix\_Commande CURRENCY, Date\_Commande DATETIME, Taille\_Commande INT, #ID\_Client); Livraison = (ID\_Livraison INT, Adresse\_initiale\_Livraison VARCHAR(50), Adresse\_finale\_Livraison VARCHAR(50), Prix\_Livraison CURRENCY, Date\_Livraison DATETIME, #ID\_Commande, #ID\_Livreur); Contient = (#ID\_Plat, #ID\_Commande);

Commande SQL :

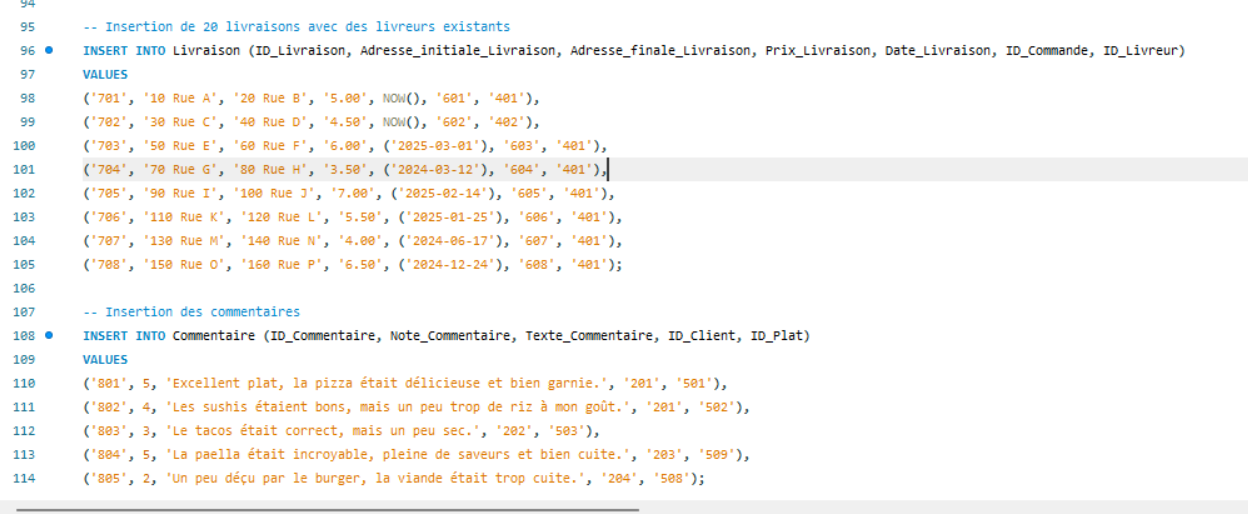
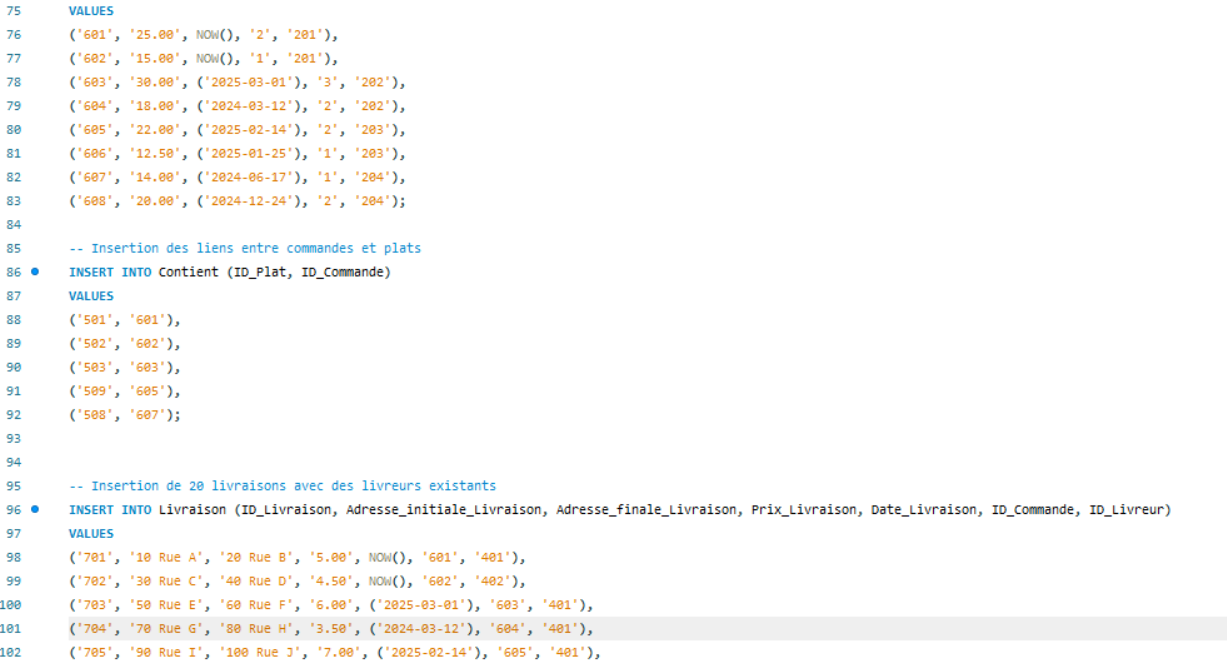
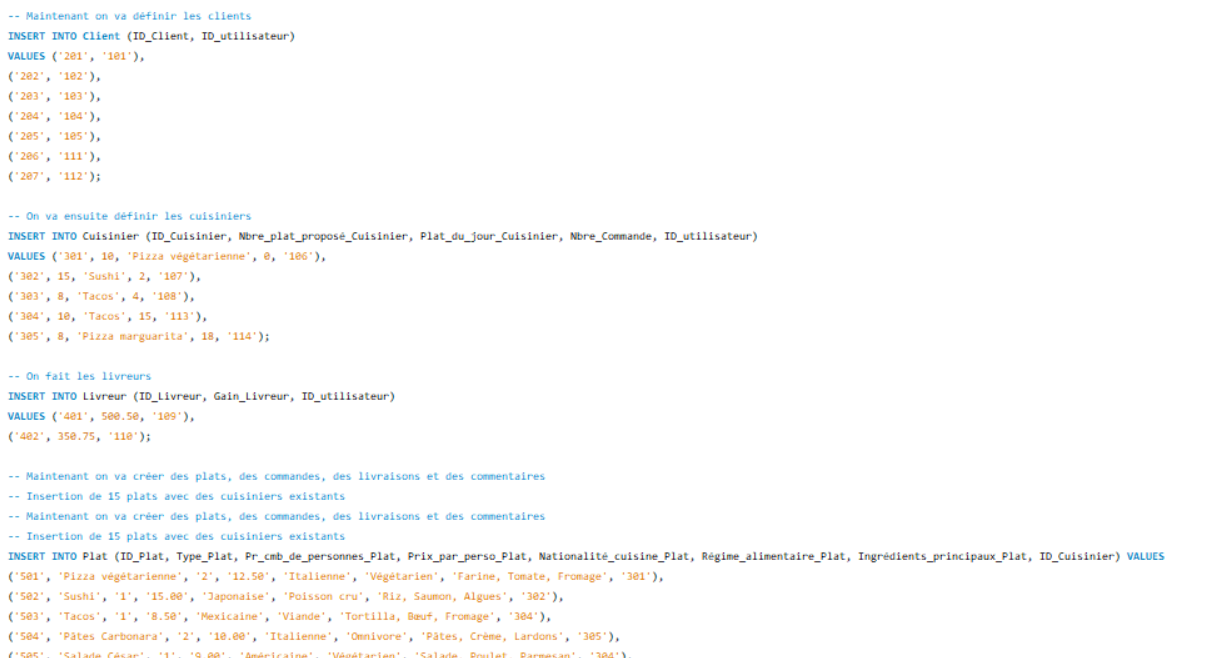
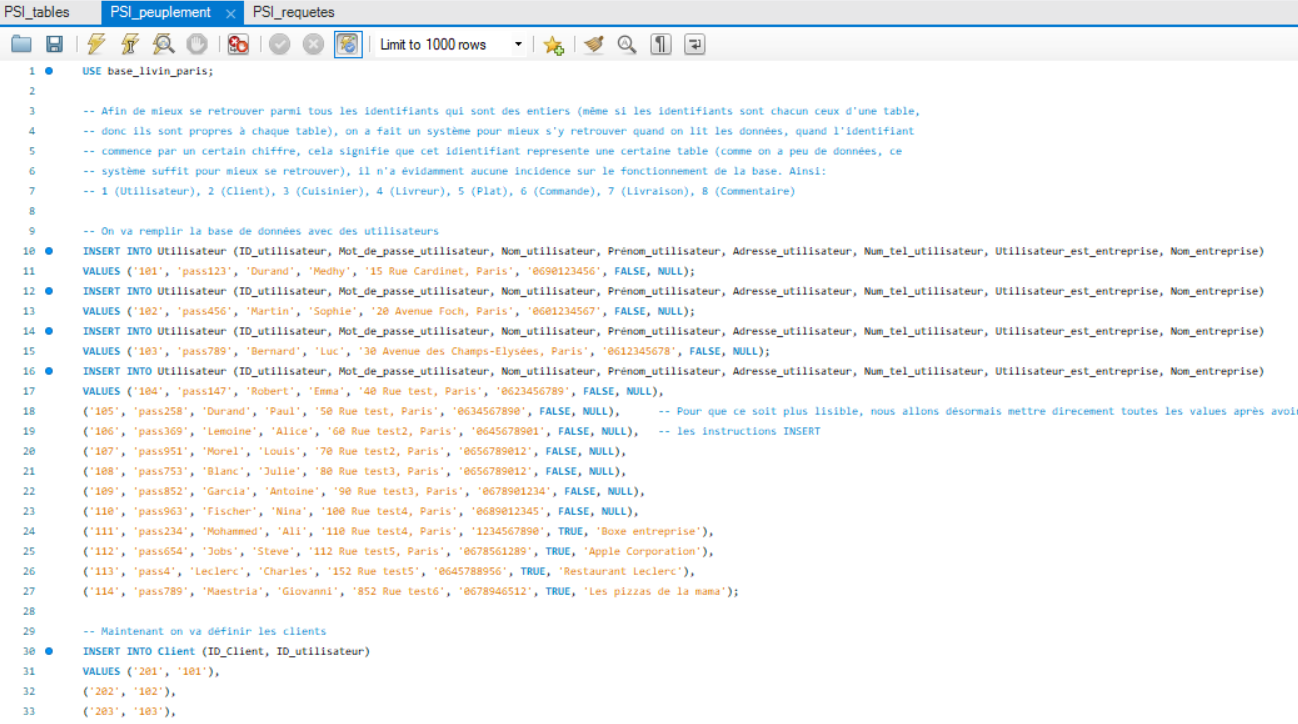
D’abord, le code générer par le looping qui créer et génère les tables SQLs pour structurer notre base de données, on a quand même dû corriger certaines parties avec l’IA pour que les tables marchent parfaitement.





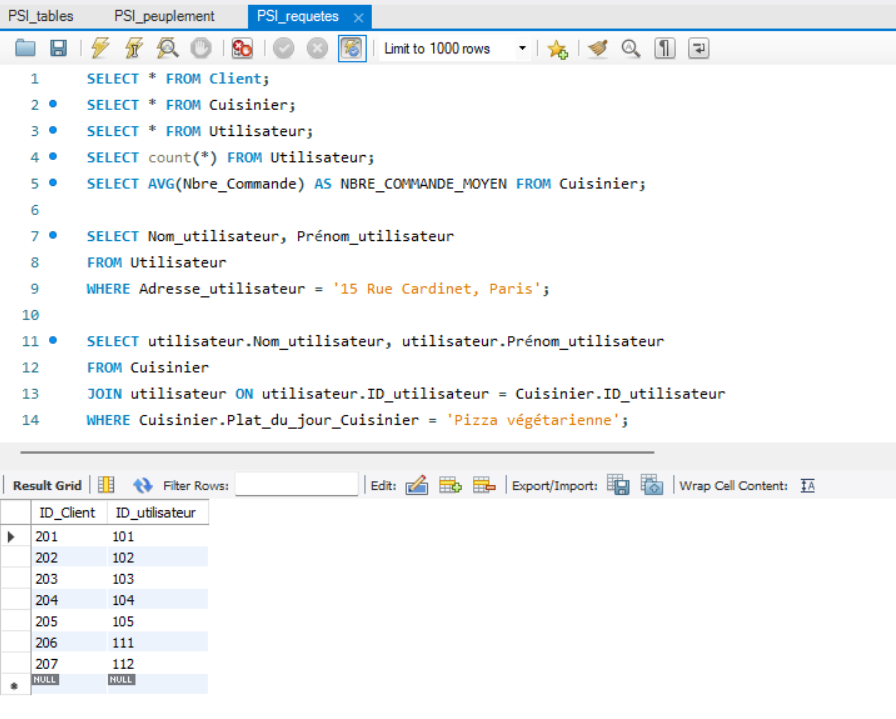
## Le Peuplement

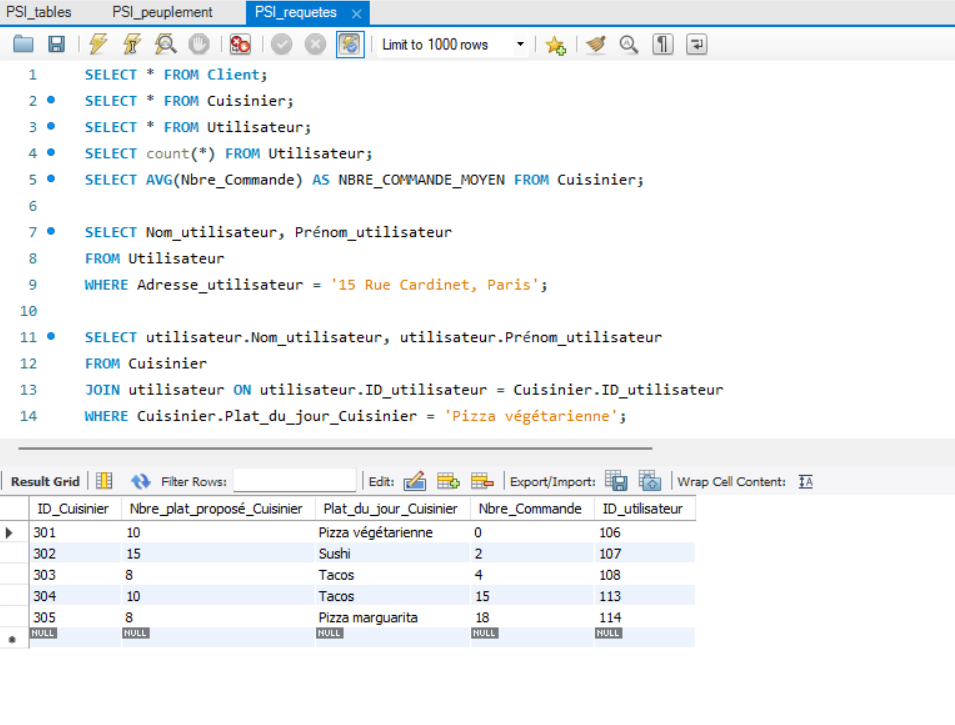
Manuellement, nous avons peuplé les bases de données. Nous n'avons pas utilisé Excel pour éviter de gérer les liens et la conversion des fichiers .xlsx en .csv. Car les documents étaient vides, il était donc plus pratique d’insérer directement ces données dans un script sql.



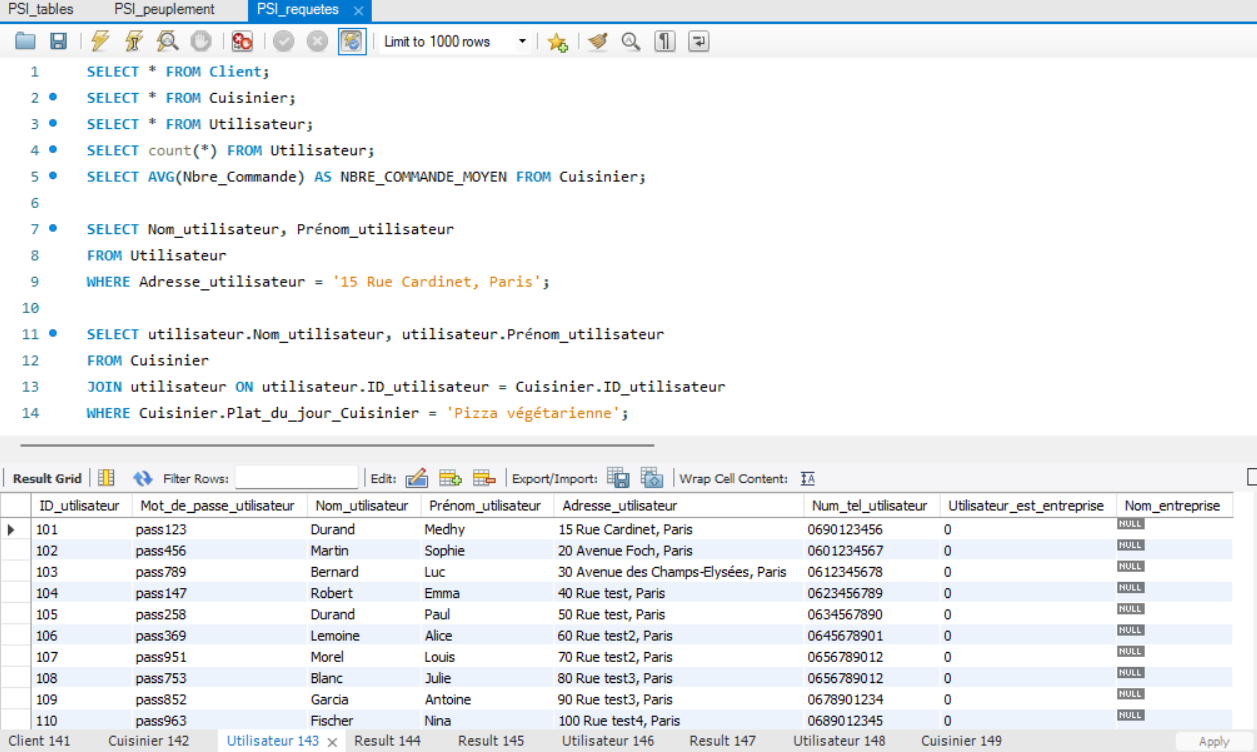
Requêtes SELECT

Requête de la ligne 1 :

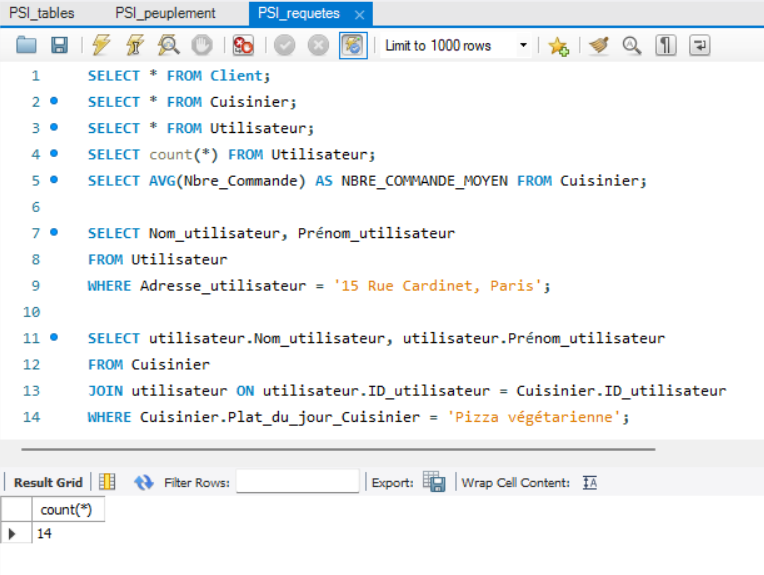


Requête de la ligne 2 :

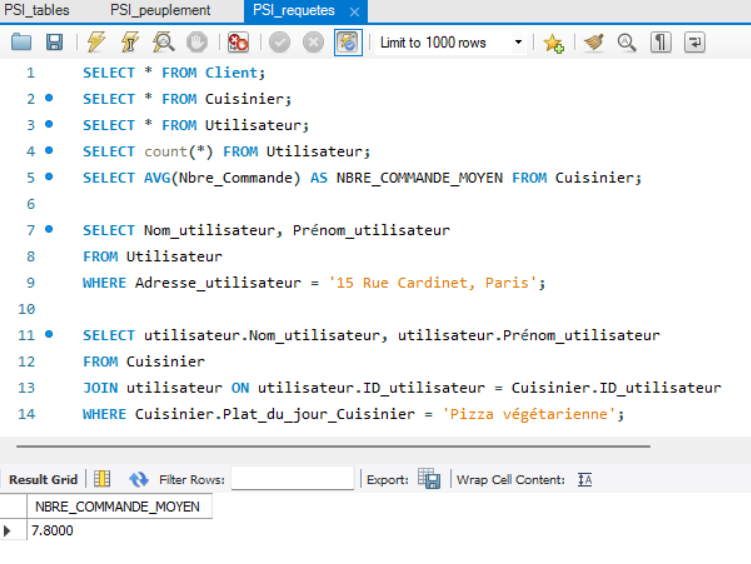
Requete ligne 3 :



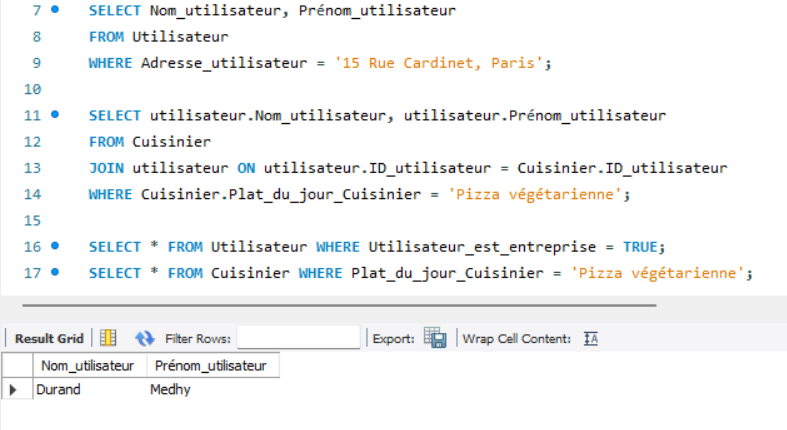
Requête ligne 4 :



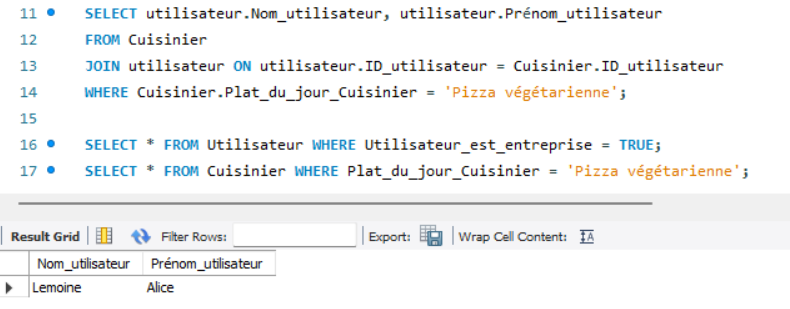
Requête ligne 5 :



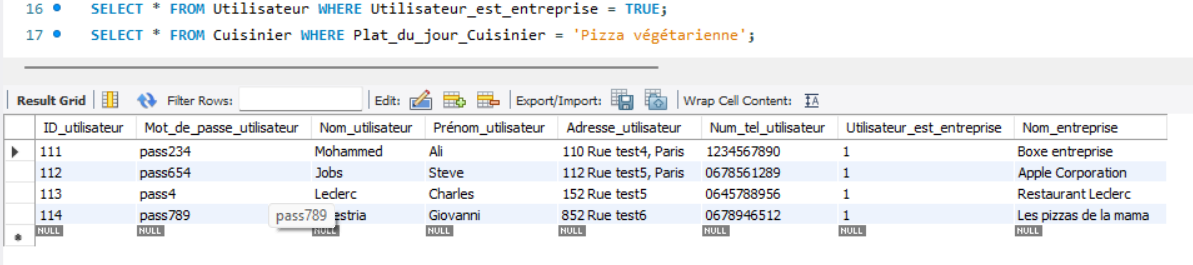
Requête ligne 7 :



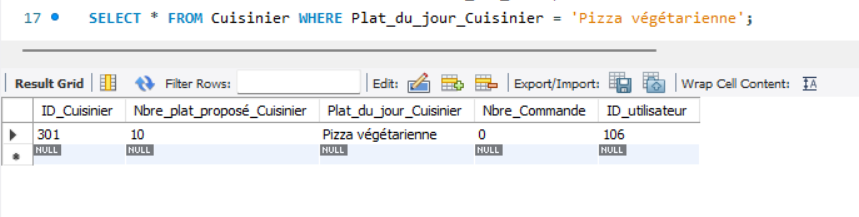
Requête ligne 11 :



Requête ligne 16 :



Requête ligne 17 :



1. Partie Graphe :

## Étape 1 : Classes Graph, Lien et Noeud

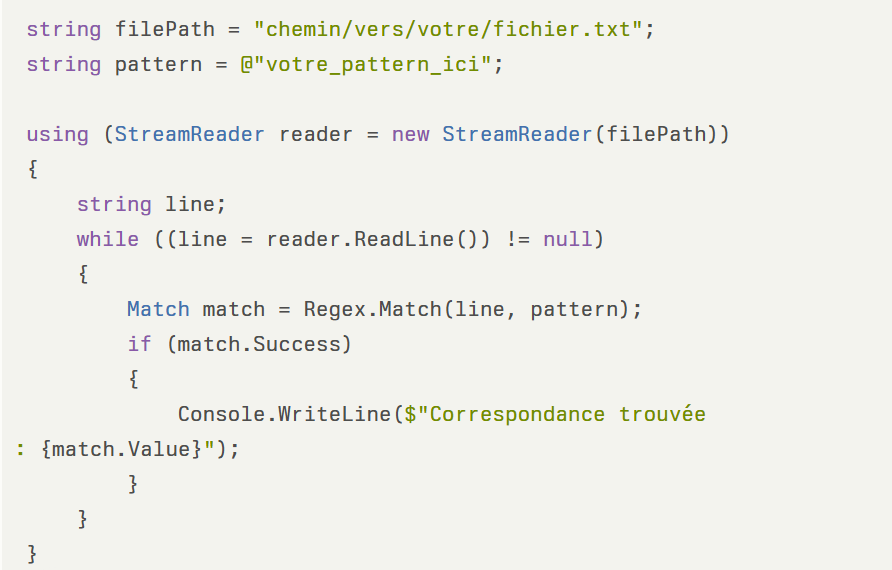
Nous commençons par définir les classes Lien et Noeud. La classe Noeud contient son nom et la liste de ses nœuds adjacents, tandis que la classe Lien définit les deux nœuds qu'elle connecte.

La classe Graph est plus complexe et comprend les attributs suivants :

* Un dictionnaire de Noeud avec le nom du nœud comme clé et le Noeud lui-même comme valeur
* Une liste de Lien regroupant tous les liens entre les nœuds
* Une matrice d'adjacence pour faciliter certaines opérations sur le graphe
* Le nombre total de nœuds dans le graphe

Le constructeur de la classe Graph extrait les informations d'un fichier texte en utilisant la classe Match de Regex. Une fois les deux nœuds formant une connexion récupérée, nous remplissons la matrice d'adjacence, le dictionnaire de Noeud (en créant de nouveaux Noeud si nécessaire) et créons le lien entre les deux.

Pour trouver comment utiliser Match on a utilisé ce prompte : “Comment retrouver un certain pattern quand on lit un fichier Txt avec C# en Utilisant la Class Match” :



## Étape 2 : Liste d'adjacence et matrice d'adjacence

La matrice d'adjacence est créée dans le constructeur de Graph. Pour la liste d'adjacence, nous utilisons la matrice d'adjacence en la parcourant ligne par ligne. Pour chaque 1 à la position [i,j] de la matrice, nous ajoutons le nœud j à la liste des nœuds adjacents au nœud i.

## Étape 3 : BFS et DFS

Les algorithmes BFS (Breadth-First Search) et DFS (Depth-First Search) sont conçus de manière similaire. Ils parcourent les nœuds adjacents jusqu'à ce qu'il n'y ait plus de nœuds à découvrir. La différence réside dans l'ordre de traitement des nœuds :

* Pour le BFS, nous utilisons une Queue (FIFO - First In First Out) pour traiter tous les nœuds adjacents avant de passer au niveau suivant.
* Pour le DFS, nous utilisons une Stack (FILO - First In Last Out) pour explorer les nœuds les plus profonds en premier.

## Étape 4 : Vérification de connexité

Pour vérifier la connexité du graph, nous appliquons une méthode basée sur la définition même de la connexité. Nous effectuons un DFS sur chaque nœud pour déterminer quels autres nœuds il peut atteindre.

Les résultats montrent que notre graphe n'est pas connexe : tous les nœuds sont connectés à 17 autres (sur un total de 20), sauf les nœuds 20 et 32 qui ne sont connectés qu'entre eux. Nous en déduisons que le graphe possède 2 composantes connexes.

## Étape 5 : Nombre de circuits

Pour compter le nombre de circuits, nous utilisons un DFS spécialisé qui s'incrémente chaque fois qu'il retrouve un nœud déjà parcouru. Nous obtenons un total de 25 circuits.

## Bonus : Caractéristiques du graphe

Notre analyse révèle que ce graphe est :

* Non orienté
* Non arborescent (présence de circuits)
* Non biparti
* Non pondéré (tous les liens ont une valeur de 1)
* Non connexe (deux composantes connexes)
* Non complet
* Non planaire
* Simple (pas de liens doublons ni de boucles)
* D'ordre 20 (nombre de nœuds)
* De taille 26 (nombre de liens)
* Sans parcours euclidien ni hamiltonien (car non connexe)

## Visualisation

Pour la visualisation, nous avons utilisé la bibliothèque Raylib, choisie pour sa flexibilité et sa richesse en fonctionnalités. Notre approche consiste à disposer les nœuds en cercle et à tracer les liens entre eux en utilisant la matrice d'adjacence.

Nous calculons les positions des nœuds à l'aide de formules trigonométriques et les stockons dans une liste de Vector2. Ensuite, nous traçons les liens entre les nœuds adjacents en utilisant les fonctions DrawLine et DrawCircle de Raylib dans une fenêtre séparée.

Cette méthode de visualisation offre une représentation claire et intuitive de la structure du graphe.