

Rapport PSI Livrable 2

Oscar MARESCHAL

Myriam KHOUJA

Juliette MOTTAY

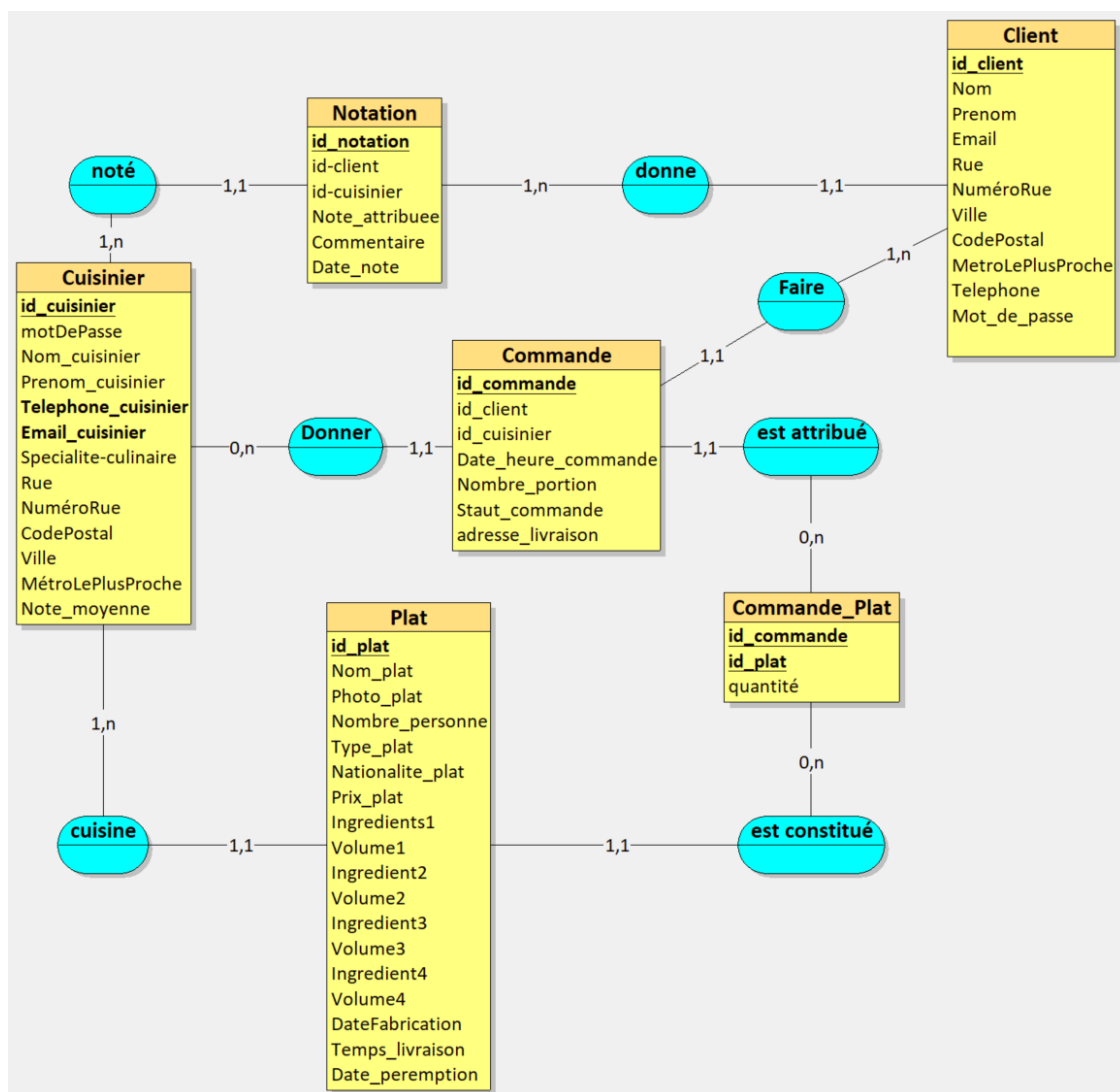
TD B

Lien GitHub :

Livrable-1---MARESCHAL-Oscar---KHOUJA-Myriam---MOTTAY-Juliette

Partie BDD :

Schéma Entité Association



Script SQL de la création de la base

```
4 • USE LivInParis;
5 • DROP TABLE commande_plat;
6 • DROP TABLE Commande;
7 • DROP TABLE Notation;
8 • DROP TABLE Client;
9 • DROP TABLE Cuisiner;
10 • DROP TABLE plat;
11 • DROP TABLE Cuisinier;
12
13 • CREATE TABLE Client (
14     id_client INT PRIMARY KEY AUTO_INCREMENT,
15     nom VARCHAR(100) NOT NULL,
16     prenom VARCHAR(100) NOT NULL,
17     email VARCHAR(100) UNIQUE NOT NULL,
18     rue VARCHAR(100),
19     numeroRue VARCHAR(10),
20     codePostal INT,
21     ville VARCHAR(100),
22     metroLePlusProche VARCHAR(100) NOT NULL,
23     telephone VARCHAR(20) UNIQUE NOT NULL,
24     mot_de_passe VARCHAR(255) NOT NULL
25 );
26
27 • CREATE TABLE Cuisinier (
28     id_cuisinier INT PRIMARY KEY AUTO_INCREMENT,
29     nom VARCHAR(100) NOT NULL,
30     prenom VARCHAR(100) NOT NULL,
31     email VARCHAR(100) UNIQUE NOT NULL,
32     telephone VARCHAR(20) UNIQUE NOT NULL,
33     rue VARCHAR(100),
34     numeroRue VARCHAR(10),
35     codePostal INT,
36     ville VARCHAR(100),
37     metroLePlusProche VARCHAR(100) NOT NULL,
38     specialite_culinaire VARCHAR(100),
39     mot_de_passe VARCHAR(255) NOT NULL,
40     note_moyenne DECIMAL(3,2) DEFAULT 0
41 );
42
```

```

43 • CREATE TABLE Plat (
44     id_plat INT PRIMARY KEY AUTO_INCREMENT,
45     id_cuisinier INT NOT NULL,
46     nom VARCHAR(100) NOT NULL,
47     photo VARCHAR(255),
48     nombre_personne INT NOT NULL,
49     type_plat VARCHAR(50) NOT NULL,
50     nationalite_plat VARCHAR(50) NOT NULL,
51     prix DECIMAL(10,2) NOT NULL,
52     ingredients TEXT NOT NULL,
53     nombre_portion INT NOT NULL,
54     date_fabrication DATE NOT NULL,
55     date_peremption DATE NOT NULL,
56     FOREIGN KEY (id_cuisinier) REFERENCES Cuisinier(id_cuisinier) ON DELETE CASCADE
57 );
58
59
60 • CREATE TABLE Commande (
61     id_commande INT PRIMARY KEY AUTO_INCREMENT,
62     id_client INT NOT NULL,
63     id_cuisinier INT NOT NULL,
64     id_plat INT NOT NULL,
65     date_heure_commande DATETIME NOT NULL,
66     nombre_portion INT NOT NULL,
67     statut_commande ENUM('En attente', 'En cours', 'Livré', 'Annulé') DEFAULT 'En attente',
68     adresse_livraison VARCHAR(255) NOT NULL,
69     FOREIGN KEY (id_client) REFERENCES Client(id_client),
70     FOREIGN KEY (id_cuisinier) REFERENCES Cuisinier(id_cuisinier),
71     FOREIGN KEY (id_plat) REFERENCES Plat(id_plat)
72 );
73 -- table pour gérer les plats commandés
74 • CREATE TABLE Commande_Plat (
75     id_commande INT NOT NULL,
76     id_plat INT NOT NULL,
77     quantite INT NOT NULL DEFAULT 1,
78     PRIMARY KEY (id_commande, id_plat),
79     FOREIGN KEY (id_commande) REFERENCES Commande(id_commande) ON DELETE CASCADE,
80     FOREIGN KEY (id_plat) REFERENCES Plat(id_plat) ON DELETE CASCADE
81 );
82
83 • CREATE TABLE Notation (
84     id_notation INT PRIMARY KEY AUTO_INCREMENT,
85     id_commande INT NOT NULL,
86     id_client INT NOT NULL,
87     id_cuisinier INT NOT NULL,
88     note_attribuee DECIMAL(2,1) CHECK (note_attribuee BETWEEN 0 AND 5),
89     commentaire TEXT,
90     date_note DATETIME DEFAULT CURRENT_TIMESTAMP,
91     FOREIGN KEY (id_client) REFERENCES Client(id_client),
92     FOREIGN KEY (id_cuisinier) REFERENCES Cuisinier(id_cuisinier)
93 );

```

Partie Algo et Graphe :

Prompts issus des IA génératives pour visualiser le graphe:

Visualisation du Graphe Métro en Utilisant des IA génératives

Exemples de prompts utilisés pour guider l'IA :

"Génère une classe en C# permettant d'afficher un graphe orienté avec sommets positionnés selon des coordonnées GPS (latitude, longitude)."

"Écris un algorithme qui lit deux fichiers CSV représentant les stations et les connexions du métro parisien, puis construit un graphe orienté prêt à être affiché dans un Form Windows."

"Ajoute la possibilité de zoomer dans la fenêtre graphique pour une exploration intuitive du graphe."

"Affiche les informations du graphe dans la console"

Résultat obtenu :

Chaque station est représentée par un sommet positionné en fonction de ses coordonnées GPS.

Les liaisons entre stations sont représentées par des arcs orientés avec flèches

L'utilisateur peut zoomer et dézoomer

☑ Toutes les données (stations, connexions) sont également affichées dans la console

Comparaison algo de chemin le plus court

- Dijkstra

L'algorithme de Dijkstra sert à trouver le plus court chemin entre un nœud de départ et tous les autres nœuds dans un graphe pondéré. Il fonctionne en sélectionnant à chaque étape le nœud non visité avec la plus petite distance estimée, puis en mettant à jour les distances de ses voisins.

Code de l' algo complet

Test : Déterminez le chemin le plus court entre deux sommets du graphe de votre choix (aux extrémités). Capture écran résultat (affichage du chemin sur le graphe plus sortie console du chemin avec changement et temps pour le cuisinier)

Dijkstra

Avantages :

- **Efficacité** : Pour les graphes avec des poids positifs, Dijkstra est très rapide, particulièrement avec une structure de tas binaire, avec une complexité de $O((V + E) \log V)$.

- **Précision** : Il trouve le plus court chemin de manière optimale en tenant compte des poids des arêtes.

Inconvénients :

- **Poids négatifs** : Il ne fonctionne pas si des arêtes ont des poids négatifs.
- **Dépendance à la structure de données** : Sa performance peut être affectée par le choix de la structure de données (tableaux simples vs tas binaires).

Bellman-Ford

Avantages :

- **Poids négatifs** : Il peut gérer les graphes avec des poids négatifs, contrairement à Dijkstra.
- **Détection de cycles négatifs** : Il peut détecter les cycles négatifs dans le graphe.

Inconvénients :

- **Complexité plus élevée** : Sa complexité est $O(V * E)$, ce qui le rend plus lent que Dijkstra, surtout pour des graphes denses.
- **Moins efficace** : Par rapport à Dijkstra, il est moins performant sur des graphes sans poids négatifs.

Floyd-Warshall

Avantages :

- **Toutes les paires de chemins** : Il calcule le plus court chemin entre toutes les paires de nœuds dans un graphe, ce qui le rend utile dans des situations où tous les chemins sont nécessaires.
- **Simplicité** : Il est relativement simple à implémenter.

Inconvénients :

- **Complexité élevée** : Sa complexité est $O(V^3)$, ce qui le rend peu adapté aux grands graphes.
- **Utilisation de la mémoire** : Il nécessite une matrice de $V \times V$ pour stocker les distances, ce qui peut consommer beaucoup de mémoire pour des graphes volumineux.

