

# iFoodDB - A Food Database

Par Romain Croughs, Chris Eid, Gabriel Goldsztajn, et Lucas Van Praag.

INFO-H-303 - Bases de données - 2023-2024

# Table des matières

## 1. iFoodDB - A Food Database

### 1. Table des matières

### 2. Introduction

### 3. Architecture de la base de données

### 4. Méthodes d'extraction des données

#### 1. Prenons l'exemple de la méthode `extract_comment`

### 5. Requêtes SQL demandées

#### 1. Requête 2

#### 2. Requête 3



#### 3. Requête 4

#### 4. Requête 5

#### 5. Requête 6

# Introduction

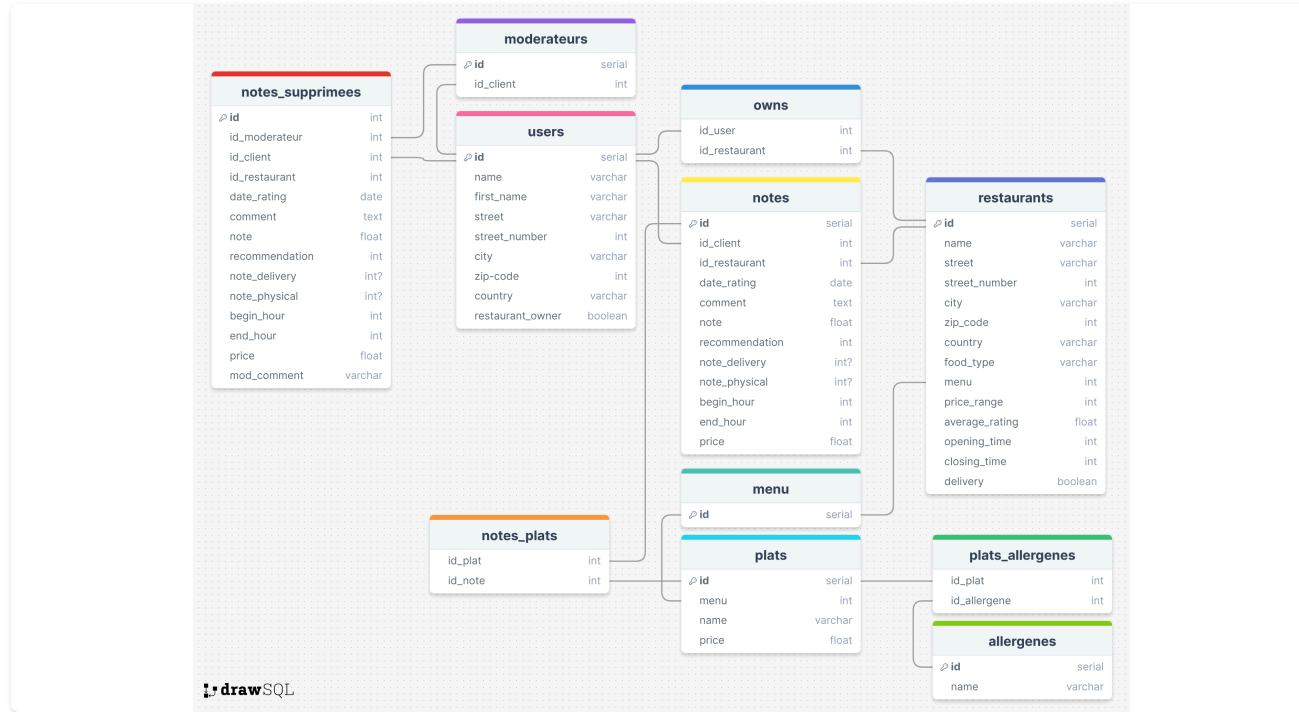
## Technologies utilisées

-  Python
-  PostgreSQL

## Scripts Python du projet

- `DDL.py` : Création des tables de base de données (n'est pas obligatoire vu que les tables sont également créés dans les autres scripts)
- `init.py` : Insertion des données dans la base de données
- `ifood.py` : CLI pour interagir avec la base de données
- `test.py` Test des requetes SQL

# Architecture de la base de données



# Méthodes d'extraction des données

- JSON: Utilisation de la librairie `json` de Python pour lire les données.
- TSV: Utilisation de la librairie `csv` de Python pour lire les données, sauf que le délimiteur est `\t`.
- XML: Utilisation de la librairie `xml.etree.ElementTree` de Python pour lire les données.

Utilisation d'une classe `Extractor` qui contient des méthodes telles que:

- `extract_restaurants(tsv_path: str) → list[Restaurant]`
- `extract_users(json_path: str) → list[User]`
- `extract_comments(xml_path: str) → list[Comment]`

Prenons l'exemple de la méthode `extract_comment`

Le TSV est le fichier le plus compliqué à parser, car il ne contient pas de balises claires comme le XML ou le JSON, nous devons donc statiquement définir les colonnes et les lire une par une.

La méthode `extract_comment` permet de parser une ligne d'un fichier TSV et de retourner un objet `Comment` .

```
def extract_comment(self, comment) → Comment:
    com = comment[0]
    note = comment[1]
    date = comment[2]
    recommendation = 0
    if comment[3] == "recommandé":
        recommendation = Recommendation.RECOMMENDED
    elif comment[3] == "déconseillé":
        recommendation = Recommendation.NOT_RECOMMENDED
    elif comment[3] == "à éviter d'urgence":
        recommendation = Recommendation.TO_BE_AVOIDED
    else:
        print(comment[3])
    restaurant = comment[4]
    noteservice = None
    notedelivery = None
    if comment[5][0] == "H":
        noteservice = int(comment[5][-1])
    else :
        notedelivery = int(comment[5][-1])

    datecomm = comment[6]
    menu = comment[7].split(';')
    price = float(comment[8])
    begin = int(comment[9])
    end = int(comment[10])
```

# Requêtes SQL demandées

## Requête 1

*SQL*

```
-- Les restaurants ayant un avis moyen de plus de 3
SELECT r.name, AVG(c.note) AS average_rating
FROM restaurants r
JOIN notes c ON r.id = c.id_restaurant
GROUP BY r.name
HAVING AVG(c.note) ≥ 3;
```

*Algèbre relationnelle*

$\pi_{\text{name}, \text{average\_rating}} \sigma_{\text{average\_rating} \geq 3}(\text{restaurants})$

*Calcul relationnel tuple*

Soit  $R$  la relation `restaurants`.

$\{r.name \mid R(r) \wedge r.average\_rating \geq 3\}$



## Requête 2

### SQL

```
-- Le restaurant avec le plat le plus cher
WITH most_expensive_dish AS ( -- Récupère le plat le plus cher par restaurant
    SELECT
        r.name AS NomRestaurant,
        p.name AS NomPlat,
        p.price
    FROM
        restaurants r,
        plats p
    WHERE
        p.menu = r.menu
    AND
        p.price = (SELECT MAX(price) FROM plats p WHERE p.menu = r.menu )
)
SELECT -- Récupère le plat le plus cher entre tous les restaurants
    med.NomRestaurant,
    med.NomPlat,
    med.price
FROM
    most_expensive_dish med
WHERE
    med.price = (SELECT MAX(med.price) FROM most_expensive_dish med);
```

## *Algèbre relationnelle*

$$a \leftarrow \pi_{\text{idR}, \text{menuR}, \text{nameR}}(\alpha_{\text{id:idR}, \text{menu:menuR}, \text{name:nameR}}(\text{restaurants}))$$

$$b \leftarrow \pi_{\text{menuP}, \text{priceP}}(\alpha_{\text{menu:menuP}, \text{price:priceP}}(\text{plats}))$$

$$c \leftarrow a \bowtie_{\text{menuP}=\text{menuR}} b$$

$$d \leftarrow \pi_{\text{menuP}, \text{priceP}}(C)$$

$$e \leftarrow c \times \alpha_{\text{PriceP:PriceP2}}(d)$$

$$f \leftarrow \sigma_{\text{PriceP} < \text{PriceP2}}(e)$$

$$g \leftarrow \sigma_{\text{menuP}=\text{menuR}}(f)$$

$$h \leftarrow \pi_{\text{name}}(g)$$

*Calcul relationnel tuple*

$r.name, p.name, p.price \mid r \in \text{restaurants} \wedge p \in \text{plats} \wedge p.menu_id = r.menu_id \wedge \nexists p_1 (p_1 \in \text{restaurants} \wedge p.price < p_1.price)$

## Requête 3

```
-- Les 10 clients ayant consommé le plus de mexicains
WITH mexican_consumption AS ( -- Récupère le nombre de fois que chaque client à mangé mexicain
    SELECT
        n.id_client,
        COUNT(*) AS mexican_count
    FROM
        notes n
        JOIN restaurants r ON n.id_restaurant = r.id
    WHERE
        r.food_type = 'mexicain'
    GROUP BY
        n.id_client
)
SELECT -- Selectionne les 10 client qui ont mang le plus de mexicain
    u.id AS user_id,
    u.name AS user_name,
    mc.mexican_count
FROM
    mexican_consumption mc
    JOIN users u ON mc.id_client = u.id
ORDER BY
    mc.mexican_count DESC
LIMIT
    10;
```

## Requête 4

```
-- Le restaurant non-asiatique proposant le plus de plats qui sont généralement proposés dans des restaurant asiatiques
WITH AsianRestaurantDishes AS(
    SELECT rd.dish_id
    FROM Restaurants r
    JOIN RestaurantDishes rd ON r.restaurant_id = rd.restaurant_id
    WHERE r.cuisine_type = 'Asian'
),
```

## Requête 5

```
-- Le code postal de la ville dans laquelle les restaurants sont les moins bien notés en moyenne
WITH average_per_restaurant AS ( -- Calcule la note moyenne de chaque restaurant
    SELECT
        r.name as NomRestaurant,
        r.zip_code as CodePostal,
        AVG(n.note) as MoyenneNote
    FROM
        restaurants r
        JOIN Notes n ON r.id = n.id_restaurant
    GROUP BY
        r.name,
        r.zip_code
),
average_per_city AS ( -- Calcule la note moyenne des restaurants pour chaque ville
    SELECT
        CodePostal,
        AVG(MoyenneNote) as MoyenneParVille
    FROM
        average_per_restaurant
    GROUP BY
        CodePostal
)
SELECT -- Finalement, sélectionne la note la plus basse dans ce classement
    CodePostal,
    MoyenneParVille
```

```
FROM
    average_per_city
WHERE
    MoyenneParVille = (SELECT MIN(MoyenneParVille) FROM average_per_city);
```

## Requête 6

```
-- Pour chaque tranche de score moyen (1/5, 2/5, 3/5, ...) de restaurant, le type de nourriture le plus représenté
WITH avg_scores AS ( -- Calcule la moyenne des notes de chaque restaurant
    SELECT
        id_restaurant,
        AVG(note) AS avg_score
    FROM
        notes
    GROUP BY
        id_restaurant
),
score_ranges AS ( -- Sépare les restaurants dans les tranches de notes associés
    SELECT
        id_restaurant,
        CASE
            WHEN avg_score ≥ 0 AND avg_score < 1 THEN 1
            WHEN avg_score ≥ 1 AND avg_score < 2 THEN 2
            WHEN avg_score ≥ 2 AND avg_score < 3 THEN 3
            WHEN avg_score ≥ 3 AND avg_score < 4 THEN 4
            WHEN avg_score ≥ 4 AND avg_score ≤ 5 THEN 5
        END AS score_range
    FROM
        avg_scores
),
```



```
restaurant_food_types AS ( -- Lie les restaurants à leur type de nourriture
    SELECT
        r.id AS restaurant_id,
        r.food_type AS type
    FROM
        restaurants r
),
food_type_counts AS ( -- Compte le nombre d'occurrence de chaque type de nourriture par catégorie de note
    SELECT
        sr.score_range,
        rt.type,
        COUNT(*) AS type_count
    FROM
        score_ranges sr
        JOIN restaurant_food_types rt ON sr.id_restaurant = rt.restaurant_id
    GROUP BY
        sr.score_range,
        rt.type
),
most_represented_type AS ( -- Récupère le type de nourriture le plus représenté par catégorie de note
    SELECT
        score_range,
        type,
        type_count,
        RANK() OVER (PARTITION BY score_range ORDER BY type_count DESC) AS rank
    FROM
```

```
SELECT
    score_range,
    type AS most_represented_type
FROM
    most_represented_type
WHERE
    rank = 1;
```