# TP19: Bases de données

On utilise la base de données construite à la fin du cours précédent, qui modélise le catalogue d'une boutique ayant ouvert dans un village, ainsi que sa liste de clients et les achats que ceux-ci ont effectué. Les prix sont donnés dans une monnaie factice appelée MPdollar. Le fichier correspondant se trouve sur Cahier de Prépa dans l'archive du TP. Les données sont adaptées de données du jeu Animal Crossing: New Horizons qui ont été recensées sur Kaggle[1]. Voici les différentes tables (on rappelle que les attributs soulignés font partie de la clé primaire):

- `villagers(id, name, species, birthday, style, color, wallpaper)` stocke les informations des villageois qui ont ouvert un compte à la boutique. Les villageois sont des animaux et ont chacun une espèce et une couleur. Cette table contient aussi le nom du papier peint de leur maison.
- `articles(id, variant_id, name, variation_name, price, color, type)` stocke les informations des différents articles du catalogue. Certains articles peuvent avoir plusieurs variations, typiquement au niveau des couleurs.
- `wallpapers(name, price, color, window_type)` stocke les informations des papiers peints proposés par la boutique.
- `referral(id_referee, id_referrer)` stocke les informations du programme de parrainage, autrement dit pour chaque client, qui l'a parrainé. Le "referee" est le parrainé, et le "referrer" est le parrain. Certains clients n'ont pas de parrain, ce qui est indiqué par la valeur NULL dans la table.
- `purchases(villager_id, article_id, variant_id, date, amount)` stocke les différents achats effectués par chaque client chaque jour. Un achat est la donnée d'un unique article acheté, potentiellement en plusieurs exemplaires, par un villageois.
- `promotions(id, article_id, month, reduction)` stocke les informations des différentes soldes réalisées par le magasin. La réduction est donnée en pourcentage enlevé au prix. Ainsi, si un enregistrement a comme valeur 70 pour la réduction, cela signifie que l'article concerné coûte seulement 30% de son prix ce mois-ci.

On fera particulièrement attention au fait que la clé primaire de la table `articles` est constituée de **deux** attributs: un identifiant principal, et un identifiant de variant (par exemple, une guitare rouge et une guitare verte auront le même identifiant principal, et des identifiants de variants différents). En particulier, lors de la jointure entre cette table et la table des achats, il bien **faire correspondre les deux attributs** dans la **condition de jointure**.

[1] kaggle.com/datasets/jessicali9530/animal-crossing-new-horizons-nookplaza-dataset

**JUSQU'A LA QUESTION 20 INCLUSE** on ne considère pas la table `promotions`.

**Q1.** Explorez les données de la base pour vous familiariser avec sa structure et avec les nombreux attributs des différentes tables. Créez un fichier `requetes.sql` pour stocker vos réponses. En SQL, les commentaires sont délimités par --:

```
-- Question 1 --
SELECT ... FROM ...
WHERE ...; -- un commentaire --
```

## Requêtes basiques

Dans cette partie, pas besoin d'utiliser `JOIN`.

**Q2.** Déterminer les prénoms des villageois rouges.

**Q3.** Déterminer l'ensemble des couleurs possibles pour les villageois.

**Q4.** Déterminer s'il existe un villageois qui a votre date d'anniversaire. Attention, les dates sont au format américain `MM-JJ` !

**Q5.** En notant $p_1, \dots p_n$ les papiers peints triés par prix croissant, donner $p_4, \dots, p_{10}$.

**Q6.** Donner les 10 articles les plus chers, sans prendre en compte les variantes (deux articles identiques mais de couleurs différentes seront comptés comme un seul article).

**Q7.** Donner les dates auxquelles un article a été acheté en exactement 7 exemplaires d'un coup.

**Q8.** Donner les dates auxquelles c'est l'anniversaire d'un villageois dont le style est "cool", mais aucun article n'a été acheté en exactement 7 exemplaires d'un coup.

## Jointures

Dans cette partie, il sera nécessaire d'utiliser `JOIN`, mais pas `GROUP BY`.

**Q9.** Décrire en français ce que fait la requête suivante:

```
SELECT v.name,
     v.color AS villager_color,
     w.color AS wallpaper_color
FROM villagers AS v
JOIN wallpapers AS w ON v.wallpaper = w.name
WHERE villager_color != wallpaper_color;
```

**Q10.** Déterminer le nom du parrain de Charlise.

**Q11.** Déterminer les villageois qui se sont référés eux-même.

**Q12.** Compter le nombre total d'achats ayant été réalisés par des villageois le jour de leur anniversaire.

**Q13.** Déterminer la liste des villageois qui ont acheté au moins un article qui n'est pas de leur couleur.

**Q14.** Déterminer les villageois n'ayant acheté aucun article le jour de leur anniversaire. Proposer une version avec `LEFT JOIN` et une version avec `EXCEPT`.

## Agrégats

**Q15.** Décrire en français ce que fait la requête suivante:

```
SELECT color, COUNT(*) AS compte
FROM villagers
GROUP BY color
```

**Q16.** Déterminer, pour chaque villageois, le nombre d'articles qu'il a acheté, l'article le plus cher et l'article le moins qu'il a acheté.

**Q17.** Déterminer pour chaque combinaison espèce-couleur-style, le nombre de villageois de cette combinaison (*on peut préciser plusieurs attributs lors d'un* `GROUP BY`). Afficher seulement les 10 combinaisons les plus courantes.

**Q18.** Déterminer le prix moyen des articles achetés par chaque villageois, et afficher les 10 villageois ayant acheté les articles les moins chers en moyenne.

**Q19.** Déterminer la somme totale dépensée par chaque villageois pendant le premier trimestre de l'année (entre le 01-01 et le 03-31). On remarquera que l'ordre par défaut en SQL sur les chaînes de caractère correspond à l'ordre chronologique des dates lorsque l'on utilise le format `MM-JJ`. Donner les 5 villageois les plus dépensiers durant cette période.

**Q20.** Reprendre la requête précédente, et n'afficher que les villageois ayant dépensé entre 40000 et 50000 MPdollars sur cette période.

## Requêtes avancées

**Q21.** L'ancêtre d'un client $A$ est le référent du référent du ... du référent de $A$. Déterminer le client dont l'ancêtre est le plus lointain. Il pourra être nécessaire de faire plusieurs requêtes pour répondre à cette question.

On considère à partir de maintenant la table des promotions. On veut déterminer combien d'argent chaque villageois a dépensé lors de l'année, promotions incluses.

Avec SQLite, on peut comparer des chaînes de caractères en utilisant l'opérateur LIKE. Il va de pair avec le symbole '`%`', que l'on appelle **joker**, et qui signifie "n'importe quoi". Par exemple, pour avoir la liste des villageois qui sont nés le 1er du mois, on peut écrire:

```
SELECT * FROM villagers v
WHERE v.birthday LIKE "%-01";
```

En SQL, l'opérateur de concaténation de chaînes de caractères se note `||`. On aurait donc aussi pu écrire:

```
SELECT * FROM villagers v
WHERE v.birthday LIKE "%" || "-" || "01";
```

**Q22.** Déterminer les achats réalisés le jour de l'anniversaire de l'acheteur alors qu'il y avait une promotion sur l'article acheté. On affichera le nom et l'anniversaire du villageois, le nom et le prix de l'article acheté, le montant acheté ainsi que le pourcentage de réduction appliqué.

On introduit une syntaxe de SQL permettant de faire des "if-then-else" sur les attributs d'une requête. Regardons la requête suivante:

```sql
SELECT DISTINCT
a.name,
CASE WHEN a.price < 5000 THEN
  "faible"
WHEN 5000 <= a.price AND a.price < 10000 THEN
  "moyen"
ELSE
  "grand"
END
FROM articles a;
```

Exécutez cette requête: elle affiche la liste des articles, avec un indicateur de leur prix comme colonne additionnel Tout le bloc entre CASE et END définit un unique attribut. On peut ainsi définir un attribut dont la valeur varie selon certaines conditions, on peut ensuite le renommer et l'utiliser comme n'importe quel attribut classique. Par exemple, pour compter le nombre d'articles dans chaque groupe de prix:

```sql
SELECT
CASE WHEN a.price < 5000 THEN
  "faible"
WHEN 5000 <= a.price AND a.price < 10000 THEN
  "moyen"
ELSE
  "grand"
END AS groupe_prix,
count(DISTINCT a.name) as nb_articles
FROM articles a
GROUP BY groupe_prix;
```

D'une manière similaire, on peut définir un nouvel attribut comme une combinaison d'autres attributs, en utilisant des opérations classiques (addition, concaténation, etc...)

Par exemple, la requête suivante affiche la liste des articles en préfixant le nom de l'article par le nom de la variation de l'article:

```sql
SELECT name, variation_name || ' ' || name as detail FROM articles;
```

**Q23.** Exécuter la requête précédente, identifier le problème qui survient pour les articles n'ayant pas de variations, et le régler en utilisant la syntaxe CASE WHEN ... END

**Q24.** Déterminer la liste de tous les achats réalisés, en ajoutant une colonne additionnelle donnant la réduction appliquée pour cet achat (0 si aucune promotion n'est en cours sur l'article acheté). *Indication: une jointure externe gauche sur les promotions peut vous permettre via les valeurs NULL de déterminer si un achat a été fait lors d'une promotion.*

**Q25.** Déterminer le villageois ayant dépensé le moins d'argent. On devra prendre en compte les promotions, mais aussi le fait que tout achat qu'un villageois réalise le jour de son anniversaire est gratuit.

**Q26.** Déterminer le villageois le moins rentable pour le magasin Nook, c'est à dire celui pour qui le ratio $\frac{\text{montant total payé}}{\text{montant théorique hors remises}}$ est minimal. **Attention:** les divisions entre entiers donnent des entiers, comme en C. Vous pouvez multiplier un nombre par `1.0` pour le forcer à devenir flottant.

**Q27.** Y a t-il des villageois n'ayant bénéficié d'aucune promotion ?