♦ TP10 – Compléments sur les bases de données relationnelles

I. — Notion de base de données relationnelle

1) Introduction

Avec l'avènement de l'informatique, un grand nombre des données qui étaient stockées auparavant sur des supports physiques ont été numérisées. De plus, l'utilisation des ordinateurs a créé de nouvelles données que l'on peut également être amené à stocker.

L'organisation et le stockage de données numériques doivent répondre à certains critères pour être efficaces afin de rendre les sources facilement modifiables et consultables :

1. éviter de stocker plusieurs fois la même information ;
2. lors d'une actualisation, éviter de devoir effectuer la même modification à plusieurs endroits ;
3. pouvoir extraire simplement une information particulière mêlée à des dizaines, des centaines voire des milliers ou des millions d'autres.

Pour répondre à ce besoin, on a développé les **bases de données** et des programmes permettant de gérer les données qu'elles contiennent de façon simple, efficace et rapide : les **systèmes de gestion de bases de données**.

Il existe différentes types de bases de données. Nous n'étudierons ici que les **bases de données relationnelles** (BDR) qui sont constituées d'un ou plusieurs tableau(x) mettant en relation différentes données, les tableaux pouvant eux-même entre mis en relation les uns avec les autres.

2) Vocabulaire

a) Tables, attributs et enregistrements

Dans une BDR, les données sont organisées dans des tableaux appelés **tables** ou **relations**.

Chaque table est identifiée par un **nom**.

Une table est composée de plusieurs colonnes et de plusieurs lignes. Dans le langage des BDR, une colonne s'appelle un **attribut** et une ligne s'appelle un **enregistrement**.

Chaque enregistrement (chaque ligne) est composé de différentes valeurs qui, comme en Python, peuvent avoir différents types (nombres entiers, nombres flottants, chaînes de caractères, booléens,...). Les éléments d'un même attribut (une même colonne) ont en revanche tous le même type. Dans le langage de BDR, le type est plutôt appelé le **domaine** de l'attribut.

Considérons la table suivante :

Produits				
Référence	Nom	Masse (en g)	En stock	Prix (en €)
554871	Confiture de fraise	250	True	4.50
554872	Confiture de fraise	500	True	6.50
543154	Pâte de coing	300	True	4.75
543155	Pâte de coing	500	True	6.75
554745	Confiture d'abricot	250	True	4.50
554746	Confiture d'abricot	500	True	6.50
562219	Gelée de pomme-rhubarbe	250	False	4.90
563557	Gelée de mûre	250	True	4.10

Le nom de cette table est *Produits*. Elle contient 8 enregistrements et 5 attributs :

- l'attribut *Référence* dont le domaine est l'ensemble des entiers ;
- l'attribut *Nom* dont le domaine est l'ensemble des chaînes de caractères ;
- l'attribut *Masse (en g)* dont le domaine est l'ensemble des entiers ;
- l'attribut *En stock* dont le domaine est l'ensemble des booléens ;
- l'attribut *Prix (en €)* dont le domaine est l'ensemble des flottants.

b) Clés

Pour identifier un enregistrement, on utilise une clé. Il en existe de deux sortes :

- Une **clé primaire** est un attribut qui prend une valeur différente pour chaque enregistrement de la table. Ainsi, la connaissance de la valeur de cet attribut permet de déterminer entièrement l'enregistrement de la table auquel il correspond.
- Lorsqu'on dispose de deux tables *table1* et *table2*, une **clé étrangère** de *table2* est une clé primaire de *table1* qui apparaît comme attribut dans *table2* (sans nécessairement être une clé primaire de *table2*). Une clé secondaire va permettre d'établir une relation entre deux tables.

Ainsi, dans la table *Produits* ci-dessus, l'attribut *Référence* est une clé primaire. Aucun des autres attributs ne peut servir de clé primaire car ils apparaissent tous dans plusieurs enregistrements différents.

Considérons une seconde table *Ventes*.

Ventes			
Numéro	Référence produit	Quantité	Mode de paiement
1	554745	2	carte
2	543154	1	espèce
3	554871	3	carte
4	563557	1	espèce
5	554871	2	chèque

L'attribut *Référence produit* est une clé étrangère de la table *Ventes* (relativement à la table *Produits*). En effet, cet attribut va permettre de lier les deux tables et, par exemple, de savoir que la vente 1 est une vente de confiture d'abricot. Ici *Référence produit* n'est pas une clé primaire de la table *Ventes* car 554871 apparaît dans plusieurs enregistrements.

On remarquera qu'il n'est pas nécessaire que la clé secondaire ait le même nom d'attribut dans les deux tables. Il faut, en revanche, que les valeurs de l'attribut soient les mêmes dans les deux tables (précisément, que toutes les valeurs de la clé étrangère de la seconde table apparaissent bien dans la clé primaire de la première table).

c) Schéma relationnel

Le **schéma relationnel** d'une table est la donnée du nom de table (en gras) suivi, entre parenthèses, de la liste de ses attributs. En général, on souligne la clé primaire et on fait précéder (ou suivre) l'éventuelle clé étrangère du symbole #.

Ainsi, le schéma relationnel de la table *Produits* est

Produits(Référence, Nom, Masse (en g), En stock, Prix (en €))

et le schéma relationnel de la table *Ventes* est

Ventes(Numéro, #Référence produit, Quantité, Mode de paiement)

II. — Langage SQL et requêtes

On se place ici dans l'optique de l'utilisateur d'une BDR (on n'abordera pas dans ce cours la création ou la modification d'une BDR). Cet utilisateur a besoin d'extraire certaines informations de la base. Pour cela, il faut utiliser un **langage de requête**. Celui que nous utiliserons est le langage SQL (abréviation de *Structured Query Language*).

À l'aide de ce langage, nous allons pouvoir exécuter des commandes, appelées **requêtes**, pour obtenir différentes données de la base à l'aide de mots-clés.

À la différence du langage Python, le SQL est insensible à la casse (i.e. on peut écrire les mots-clés indifféremment en majuscules ou en minuscules). Cependant, l'usage est d'écrire les mots-clés en majuscules. Il est également insensible à l'indentation et au retour à la ligne (une même instruction peut être écrite indifféremment sur une seul ligne ou sur plusieurs lignes). Comme en Python, les chaînes de caractères doivent être encadrées par des apostrophes ou des guillemets et, à l'intérieur d'une chaîne de caractères, la casse est importante (autrement dit, la chaîne `'confiture'` est différente de la chaîne `'Confiture'`). Lorsque la chaîne de caractères contient elle-même des espaces et/ou des apostrophes, il est impératif d'utiliser des guillemets (par exemple, on écrit `"condifure d'abricot"` et non pas `'confiture d'abricot'`).

1) Projection

La **projection** est une requête qui consiste à extraire une (ou plusieurs) colonne(s) de la table. Elle s'effectue à l'aide du mot-clé `SELECT`.

Reprenons la table *Produits* vue ci-dessus. Si on souhaite obtenir la table contenant seulement les références et les noms des produits, on écrit

```
SELECT Référence,Nom FROM Produits
```

Pour obtenir toutes les colonnes de la table *Produits*, on écrit

```
SELECT * FROM Produits
```

Il est possible, lors de la projection, de renommer un attribut grâce au mot-clé `AS`. Ainsi,

```
SELECT Référence AS 'Ref', Nom AS 'Nom du produit'
FROM Produits
```

renvoie les colonnes *Référence* et *Nom* de la table *Produits* mais en les renommant respectivement *Ref* et *Nom du produit*.

Si on souhaite obtenir les différents types de produits, on peut utiliser

```
SELECT Nom FROM Produits
```

Cependant, on obtiendra plusieurs fois *Confiture de fraise*, *Pâte de coing* et *Confiture d'abricot*. Si on veut éviter les doublons, on peut utiliser le mot-clé `DISTINCT` comme suit :

```
SELECT DISTINCT Nom FROM Produits
```

Lors d'une projection, on peut également ordonner les données d'une colonne par ordre croissant ou décroissant (ordre des nombres pour les entiers et les flottants, ordre alphabétique pour les chaînes de caractères) à l'aide des mots-clés `ORDER BY ... ASC` ou `ORDER BY ... DESC`.

Ainsi, si on veut ordonner la table *produit* dans l'ordre des références croissantes, on utilise la requête

```
SELECT * FROM Produits ORDER BY Référence ASC
```

2) Sélection

La **Sélection** est une requête qui consiste à extraire une (ou plusieurs) ligne(s) de la table. Elle s'effectue à l'aide du mot-clé `WHERE` ajouté en complément de `SELECT` et suivie d'une ou plusieurs conditions. Ces conditions peuvent utiliser

- des comparaisons (sur des nombres ou des chaînes de caractères) : `=` (est égal à), `!=` (est différent de), `>=` (est supérieur ou égal à), `<=` (est inférieur ou égal à), `>` (est strictement supérieur à), `<` (est strictement inférieur à) ;
- des connecteurs logiques avec les mots-clés `OR` (ou), `AND` (et), `NOT` (négation) ;
- un test d'appartenance à une liste de valeurs avec le mot-clé `IN`. Attention, en SQL, une liste s'écrit entre parenthèses ;
- un test d'apparition d'une chaîne de caractères dans une autre avec le mot-clé `LIKE`.

Ainsi, pour obtenir tous les produits de la table *Produits* dont la masse est 250 grammes, on utilise la requête

```
SELECT * FROM Produits WHERE "Masse (en g)" = 250
```

Remarque. On notera dans la requête `WHERE` l'utilisation indispensable des guillemets (et non pas des apostrophes) quand le nom de l'attridut contient des espaces ou des apostrophes.

Pour obtenir la liste des produits ayant une masse supérieure ou égale à 300 grammes et un prix inférieur ou égal à 6 €, on écrit

```
SELECT Nom FROM Produits
WHERE "Masse (en g)" >= 300 AND "Prix (en €)" <= 6
```

Pour extraire les ventes par espèce ou par chèque de la table *Ventes*, on peut utiliser la requête

```
SELECT * FROM Ventes
WHERE "Mode de paiment" IN ('espèce', 'chèque')
```

Pour le même résultat, on peut aussi utiliser

```
SELECT * FROM Ventes
WHERE "Mode de paiment" != 'carte'
```

Pour extraire de la table *Produits* seulement les confitures (sans les pâtes et les gelées), on utilise la requête

```
SELECT * FROM Produits WHERE nom LIKE '%Confiture%'
```

Remarque. On notera les `%` qui entourent la chaîne de caractères et qui signifient qu'on cherche la chaîne `Confiture` précédée et suivie de n'importe quelles autres chaînes de caractères. Si on voulait chercher les chaînes de caractères qui commencent par `Confiture`, on écrirait `'Confiture%'` et, pour les chaînes qui se terminent par `Confiture`, `'%Confiture'`.

Pour finir ce paragraphe, notons qu'on peut dans les projections et les sélections effectuer des calculs avec les opérateurs habituels : `+`, `-`, `*`, `/`.

Exercice 1. Se connecter à NetO'Centre puis ouvrir Capytale. Accéder ensuite à l'activité de l'exercice 1 avec la référence : **b5e4-3135809**.

Exercice 2. Accéder à l'activité de l'exercice 2 avec la référence : **c013-3136348**.

3) Jointure

Lorsque deux tables *table1* et *table2* d'une même base de données ont un attribut en commun (sans pour autant que cet attribut ait le même nom dans les deux tables), on peut effectuer la **jointure** des deux tables selon cet attribut, ce qui revient en quelque sorte à « coller » les deux tables l'une à l'autre en utilisant l'attribut qu'elles ont en commun.

Reprenons la table *Produits* et supposons qu'on dispose d'une autre table *Stock* de la forme suivante :

Stock	
Ref	Nombre
554871	34
554872	21
543154	17
543155	12
554745	30
554746	24
562219	0
563557	3

L'attribut *Référence* de la table *Produit* et le même que l'attribut *Ref* de la table *Stock*. On va donc pouvoir effectuer une jointure de ces deux tables selon cet attribut. La syntaxe est la suivante :

```
SELECT * FROM Produits JOIN Stock
ON Produits.Référence = Stock.Ref
```

On obtiendra alors la table

Référence	Nom	Masse (en g)	En stock	Prix (en €)	Nombre
554871	Confiture de fraise	250	True	4.50	34
554872	Confiture de fraise	500	True	6.50	21
543154	Pâte de coing	300	True	4.75	17
543155	Pâte de coing	500	True	6.75	12
554745	Confiture d'abricot	250	True	4.50	30
554746	Confiture d'abricot	500	True	6.50	24
562219	Gelée de pomme-rhubarbe	250	False	4.90	0
563557	Gelée de mûre	250	True	4.10	3

Remarque. On peut opérer une jointure en particulier selon une clé étrangère de la seconde table qui est une clé primaire dans la première. Cependant, on peut opérer des jointures sur des attributs qui ne sont pas des clés. Par ailleurs, il n'est pas nécessaire que les valeurs dans les deux attributs soient dans le même ordre et il est possible que des mêmes valeurs dans l'attribut de la seconde table apparaissent plusieurs fois.

Si on trouve que les noms des tables sont trop longs, on peut utiliser des alias pour raccourcir la syntaxe. Ainsi, la requête précédente peut aussi s'écrire :

```
SELECT * FROM Produits AS T1 JOIN Stock AS T2
ON T1.Référence = T2.Ref
```

On peut combiner la jointure avec la projection et la sélection. Ainsi, par exemple,

```
SELECT T1.Nom, T1."Masse (en g)", T2.Nombre
FROM Produits AS T1 JOIN Stock AS T2
ON T1.Référence = T2.Ref
WHERE T1."Masse (en g)" = 250 AND T2.Nombre > 0
```

renvoie les noms des produits de 250 grammes avec les stocks correspondants, lorsque le produit est encore en stock :

Nom	Masse (en g)	Nombre
Confiture de fraise	250	24
Confiture d'abricot	250	30
Gelée de mûre	250	3

4) Opérateurs d'agrégation

Lorsque les éléments d'un attribut sont des nombres (entiers ou flottants), on peut effectuer différentes opérations sur l'ensemble des valeurs de la colonne. Plus précisément,

- L'opérateur `COUNT(att)` permet de déterminer le nombre d'éléments de l'attribut `att`. Cet opérateur compte plusieurs fois un même élément s'il est répété. Pour obtenir, le nombre d'éléments différents dans l'attribut `att`, on utilise `COUNT(DISTINCT att)`. Enfin, `COUNT(*)` renvoie le nombre total d'enregistrements de la table.
- Les opérateurs `MIN(att)` et `MAX(att)` renvoient respectivement le minimum et le maximum des éléments de l'attribut `att`.
- Les opérateurs `SUM(att)` et `AVG(att)` renvoient respectivement la somme et la moyenne des éléments de l'attribut `att`.

Ainsi, par exemple, si on reprend la table *Produits*

```
SELECT COUNT(Référence) FROM Produits
```

renvoie le nombre de références différentes (8 en l'occurence) alors que

```
SELECT COUNT(DISTINCT Nom) FROM Produits
```

renvoie le nombre de types de produits différents (à savoir 5).
Pour obtenir le prix moyen des produits de la table, on utilise

```
SELECT AVG("Prix (en €)") FROM Produits
```

Remarque. Il est possible de combiner des opérations d'agrégation et des opérations arithmétiques sur les attributs. Par exemple, si on souhaite calculer la moyenne de la somme des valeurs de l'attribut `att1` et de celles de l'attribut `att2`, on utilise `AVG(att1+att2)`.

Il peut parfois être intéressant de regrouper des enregistrements pour les compter ensemble. Par exemple, si on veut savoir le prix moyen d'un pot de 250 grammes, d'un pot de 300 grammes et d'un pot de 500 grammes, on va devoir regrouper les produits ayant le même poids. Cela peut se faire à l'aide du mot-clé `GROUP BY` de la manière suivante :

```
SELECT "Masse (en g)", AVG("Prix (en €)") AS 'Prix
   moyen'
FROM Produits GROUP BY "Masse (en g)"
```

qui renvoie

Masse (en g)	Prix moyen
250	4.5
300	4.75
500	6.583333333333333

On peut vouloir combiner une fonction d'agrégation avec groupement et une sélection. Imaginons, par exemple, qu'on souhaite connaître les produits dont le stock total est supérieur ou égal à 20 pots (sans se préoccuper de savoir le poids des pots). Si on utilise la requête

```
SELECT Nom, SUM(Nombre) FROM Produits AS T1
JOIN Stock AS T2 ON T1.Référence = T2.Ref
WHERE Nombre >= 20 GROUP BY Nom
```

la sélection va se faire avant le groupement, ce qui va éliminer les pots de pâte de coing alors qu'il reste $17 + 12 = 29 > 20$ pots. On pourrait être tenté de placer la sélection `WHERE SUM(Stock)>=20` après `GROUP BY` mais ce n'est pas licite en SQL. Pour contourner ce problème, il faut utiliser le mot-clé `HAVING` de la manière suivante :

```
SELECT Nom, SUM(Nombre) FROM Produits AS T1
JOIN Stock AS T2 ON T1.Référence = T2.Ref
GROUP BY Nom HAVING SUM(Nombre)>=20
```

qui renvoie

Nom	SUM(Nombre)
Congiture d'abricot	54
Confiture de fraise	55
Pâte de coing	29

5) Requêtes imbriquées

Il est possible d'utiliser une requête à l'intérieur d'une autre requête. On parle alors de **requêtes imbriquées**.

Imaginons, par exemple, qu'on souhaite connaître les produits dont le prix est supérieur ou égal à la moyenne.

Pour cela, on va utiliser une requête qui calcule le prix moyen puis une autre requête pour renvoyer les noms des produits dont le prix est supérieur à cette moyenne. On peut le faire avec le code suivant

```
SELECT * FROM Produits
WHERE "Prix (en €)" > (
  SELECT AVG("Prix (en €)") FROM Produits )
```

Remarque. Une projection renvoyant une table, on peut également écrire une requête imbriquée sous la forme `SELECT ... from (SELECT ...)`.

Exercice 3. Accéder à l'activité de l'exercice 3 avec la référence : **5f36-3142222**.

Exercice 4. Accéder à l'activité de l'exercice 4 avec la référence : **8f87-3144593**.

Exercice 5. Accéder à l'activité de l'exercice 5 avec la référence : **a87f-3145324**.

Exercice 6. Accéder à l'activité de l'exercice 6 avec la référence : **c96d-3147665**.