1 Introduction

Grâce l'augmentation du débit des réseaux internet et l'informatisation de nombreux objets de notre quotidien, la récolte de données informatiques devient de plus en plus importante. La création de bases de données et leur exploitation sont possibles.

Certaines bases de données peuvent atteindre des tailles de plusieurs téraoctets en mémoire. Utiliser alors les structures classiques vue en informatique (listes, dictionnaires...) peut ne pas suffire à obtenir de bonnes performances.

On utilise donc des outils spécialisés dans le traitement des données. Nous verrons le SQL (Structured query language) dans ce cours.

1.1 Exemple

Je souhaite faire un programme informatique pour pouvoir suivre les livres que je prête à mes amis, car je n'arrive plus à suivre qui m'emprunte quoi et ce que je peux lui suggérer à lire ensuite.

Ma première idée est de faire un fichier Excel où je note ligne par ligne les détails de ce que je prête à mes amis. Voici un exemple de ce que donne un tel fichier :

Titre	Auteur	Éditeur	Langue	Nom Emprunteur	date_emprunt	est_revenu
Harry Potter	J. K. Rowling	Gallimard	Français	Max	20230801	OUI
L'informatique en CPGE	E Cochard, S Rainero	Ellipse	Français	Mo	20230901	NON
Harry Potter	J. K. Rowling	Gallimard	Français	Li	20230905	OUI
Harry Potter	J. K. Rowling	Bloomsbury	Anglais	Jenny	20230906	OUI
La vie de la forêt	Sarah Loulendo	Sens Dessus Dessou	Français	Max	20230910	NON
Carnet	Marcel Proust	Gallimard	Français	John	20230909	NON

1) Quels défaut en terme d'efficacité pouvez vous faire à cette liste de stockage.

Pour stocker efficacement ces tables, on va essayer de séparer les informations qui sont redondantes.
On propose donc de faire plusieurs tables pour séparer nos données :

- Une pour les livres possédés
- Une pour les auteurs
- Une pour les éditeurs
- Une pour les langues
- Une pour les amis
- Une pour les emprunts

A chacun des auteurs, je vais attribuer un identifiant unique. Il me permettra d'identifier chaque auteur de manière unique, sans avoir à restocker toutes ses informations.
Je fais de même pour chacune des autres tables. On appellera cet identifiant **clef primaire**.

Pour communiquer efficacement, on va voir le vocabulaire des bases de données.

2 Schéma entité-association

2.1 Vocabulaire

- **Table ou relation** : Une table est un ensemble de données cohérente. Une table est définie par son nom et ses attributs. *Ex : La table **auteur***
- **attribut ou colonne** : Un attribut est une colonne d'une table. Il représente une composante de l'information stocké. *Ex : Les prénoms pour la table **auteur***
- **enregistrement, ligne ou n-uplet** : Un n-uplet est une ligne d'une table. Elle contient pour chaque attribut des valeurs. *1, J.K. Rowling, 31/07/1965 est une ligne de la table auteur.*
- **clef primaire** : Une clef primaire est un ensemble d'attribut pour une table données tel que si deux lignes possèdent les mêmes valeurs pour ces attributs, alors ce sont les mêmes lignes. Le plus souvent, c'est un entier donnée de manière unique dont le seul but est d'identifier une ligne. Il est obligatoire d'avoir une clef primaire par table.
- **clef étrangère** : Une clef étrangère est un ensemble d'attribut d'une table qui fait référence à la clef primaire d'une autre table. Si une ligne d'une table A possède des valeurs $(v_1, ..., v_n)$ sur une clef étrangère faisant référence à une table B, alors il existe une ligne de la table B possédant les valeurs $(v_1, ..., v_n)$ sur la clef primaire. *Ex : La table **Livre** peut avoir un attribut entier qui fait référence à l'identifiant de la table **auteur***
- **domaines** : Le domaine d'un attribut est l'ensemble de valeurs que peut prendre un attribut. *Ex : L'âge d'un auteur est un **entier***. Dans le programme, on se restreint aux entiers, flottants et chaînes.

- **Schéma de table de données** : Contient toutes les informations et règles qui régissent les tables les unes entre les autres, notamment les contraintes de clef primaire et de clef étrangère pour les attributs.

2.2 Schéma

Le type de schéma que l'on utilise pour décrire la situation modélisée est le schéma entité-association.

On construit le schéma de cette manière :

- On choisit tout d'abord les entités de notre schéma, que l'on place dans des rectangles avec leurs noms. Ce sont les entités qui jouent un rôle dans le réel que l'on souhaite stocké.
- On ajoute ensuite à chaque entité les attributs à stocker qui sont nécessaire à la bonne description de notre modèle. Dans le cas de notre exemple, cela peut nous donner cinq ou six entité :
  - Table auteurs : id : INT, prénom : CHAR, nom : CHAR, date_naissance : CHAR
  - Table éditeurs : id : INT, nom_editeur : CHAR
  - Table langue : nom_langue : CHAR (on estime que le nom d'une langue est unique)
  - Table livre : ISBN : CHAR, titre : CHAR A_ecrit lien 1-* 0-* auteurs, A_editer lien 1-1 0-* editeur, en_tel_langue lien 1-1 0-* langue
  - Table ami : id : INT, prénom : CHAR
  - (Table emprunts : id : INT, date_emprunt : CHAR, est_rendu : BOOL lien est_emprunte 1-1 0-* livre, lien a_emprunte 1-1 0-* amis))
- On relie ensuite les entités qui ont des liens parce qu'on appelle des relations. On écrit le nom de ces relations dans des losanges. Dans le cas de notre exemple, les emprunts peuvent être vu comme une entité, car c'est le but de notre modélisation, de pouvoir stocker et suivre les emprunts, mais les emprunts peuvent aussi être vu comme une relation, qui relie livres et amis. Un schéma entité-association est un moyen de communication. Comme il existe deux phrases transmettant le même message, il peut exister deux schéma décrivant exactement la même situation.
- On peut éventuellement rajouter des attributs aux relations, et écrire le nom des rôles de chaque entité dans la relation. Par exemple, pour la relation emprunt, le livre a un rôle d'emprunté, et l'ami le rôle d'emprunteur. Souvent, on préfère écrire des rôles quand cela est nécessaire à la compréhension de la relation.
- On rajoute les cardinalités des relations. Une relation R qui relie deux tables A et B ayant x-y côté A indique qu'un élément de a est relié au minimum à x éléments de B et au maximum à y éléments de B. Dans le cas où il n'y a pas de maximum, on écrit * ou n.

Ne pas effacer le schéma précédent. Prévenir de garder un peu de place

Dans le schéma entité-association, on écrit chaque table de notre modélisation dans un rectangle, où l'on écrit le nom. En dessous du nom, on liste les attributs, et on souligne tous les éléments qui font parties de la clef primaire d'une table.

On relie ensuite les tables qui se font lien entre elles. La table auteur est en lien avec la table livre car les livres sont écrits par des auteurs. On note la cardinalité x,y de la relation, et on peut noter le nom de la relation. Dans les cas où le nom de la relation peut paraître évident, on est pas obligé de la noter.

Si on relie la table A à la table B en notant 1-4, 0-3, cela veut dire qu'une entité de la table A est en lien avec au moins 1 entité de la table B et au maximum 4 entités. La table B elle est en lien avec au minimum 0 entité de la table A et au maximum 3 entités de la table B. On écrit * au maximum quand il n'y a pas de limite au nombre de relation faisable.

2) Dessiner le schéma entité association pour la situation suivante :

On souhaite modéliser un réseau social. Il nous faut donc pouvoir stocker les utilisateurs, qui possède un pseudonyme, un identifiant de compte et une année de naissance.

Les utilisateurs sont capables d'être amis entre eux, et ils peuvent poster des publications. Les publications possèdent un identifiant, un texte et une date de publication.

Les publications peuvent être commentées et les commentaires peuvent être commentés par d'autres commentaires.

A remarquer : Ici, la relation ami relie deux fois la même table. Il n'y a pas besoin d'écrire de rôle car ils sont clair. Deux possibilité aussi, une table pour les commentaires, ou séparer commentaire de publications ou commentaire de commentaires.

2.3 Passer du schéma entité-association au schéma de tables

On souhaite modéliser nos relations, pour savoir comment les stocker dans notre base de données en pratique.

Pour cela, les associations de types 1-1 table A et 1-* table B peuvent être représentées par une clef étrangère dans la table A.

Exemple : Dans la table livre, on va ajouter les attributs suivants id_editeur : INT, id_langue : INT. Pour la table emprunts, on va ajouter les attributs id_amis : INT, id_livre : INT

Pour les associations 1-*, 1-*, on doit créer une nouvelle table qui liste les relations existante.

Exemple : Pour la relation A_ecrit, on doit créer une table qui possède deux attributs étant des clefs étrangères : id_auteur et id_livre. La clef primaire de cette table est le couple (id_auteur,id_livre)

On peut alors faire le schéma de table, qui diffère un peu par rapport au schéma entité-association :
Rajouter attribut plus relation transformer en table

Différence : On relie les clefs étrangères aux tables dont elles font référence.
Maintenant que l'on pu écrire un schéma de bases de données, on est organisé pour implémenter sur ordinateur notre schéma de stockage. Pour cela, on va utiliser un nouveau langage, le SQL (Structured query language).

3 SQL

3.1 Introduction

Pour stocker informatiquement nos bases de données, on utilise des logiciels appelés SGBD (système de gestion de bases de données). Dans notre cours, on va utiliser le SGBD donné par le logiciel Capytale, sur l'ENT. Le langage SQL utilisé par les SGBD peuvent différer sur des détails, mais est globalement le même.

Dans le programme de CPGE, on se limite à effectuer des requêtes sur des bases de données déjà existante, on ne va pas apprendre à créer des tables ou à insérer de nouvelles lignes dans des tables existantes.

Pour introduire les instructions de SQL, on va utiliser cette instance de base de données :

id	Prenom	nom	annee_naissance
1	J .K	Rowling	1965
2	Etienne	Cochard	1970
3	Sophie	Rainero	1976
4	Sarah	Loulendo	1983
5	Marcel	Proust	1871
6	Albert Cohen	Cohen	1895

Table Auteur

id	Prenom
1	Max
2	Li
3	Mo
4	Jenny
5	John
6	Sarah

Tables Amis

id	nom
1	Gallimard
2	Ellipse
3	Bloomsbury
4	Sens dessus Dessous

Tables Editeur

nom
Français
Anglais

Table Langue

id	titre	id_editeur	Langue
1	Harry potter	1	français
2	Harry potter	3	anglais
3	Mangeclous	1	français
4	L'informatique en CPGE	2	français
5	La vie de forêt	4	français
6	Carnet	1	français

Table Livre

id_livre	id_auteur
1	1
2	1
3	6
4	2
4	3
5	4
6	5

Table A_ecrit

id_emprunt	id_livre	id_amis	date	est_rendu
1	1	1	20230801	True
2	4	2	20230901	False
3	1	3	20230905	True
4	2	4	20230906	True
5	5	1	20230910	False
6	6	5	20230909	False

Table Emprunt

3.2 Requête de bases

```
SELECT * FROM Auteur
```

id	Prenom	nom	annee_naissance
1	J .K	Rowling	1965
2	Etienne	Cochard	1970
3	Sophie	Rainero	1976
4	Sarah	Loulendo	1983
5	Marcel	Proust	1871
6	Albert Cohen	Cohen	1895

Une requête sera toujours composée des mots clefs SELECT et FROM. L'ordre des mots clefs que l'on présentera sera aussi toujours le même, il est impossible d'échanger l'ordre de deux mots clefs. L'exécution, contrairement au python, ne se fait pas forcément de manière séquentielle

```
SELECT id, nom FROM Auteur
```

id	nom
1	Rowling
2	Cochard
3	Rainero
4	Loulendo
5	Proust
6	Cohen

Derrière SELECT, on écrit les colonnes que l'on souhaite voir dans notre résultat. On sépare chaque colonne par une virgule. Derrière FROM, on écrit la table que l'on souhaite utiliser pour effectuer notre requête. Vu à la requête précédente, on utilise * pour sélectionner tous les attributs possibles.

```
SELECT id AS identifiant, nom
FROM Auteur
```

identifiant	nom
1	Rowling
2	Cochard
3	Rainero
4	Loulendo
5	Proust
6	Cohen

AS permet le renommage. Il fonctionne sur les noms des colonnes mais aussi sur les noms de tables.

```
SELECT prenom, nom, 2024 - annee de naissance AS age
FROM Auteur
```

Prenom	nom	age
J .K	Rowling	59
Etienne	Cochard	54
Sophie	Rainero	48
Sarah	Loulendo	41
Marcel	Proust	153
Albert Cohen	Cohen	129

SQL permet de réaliser des opérations sur les entiers. Les opérateurs sont +,-,*,/.

```
SELECT prenom, nom, 2024 - annee de naissance AS age
FROM Auteur WHERE age >= 100
```

Prenom	nom	age
Marcel	Proust	153
Albert Cohen	Cohen	129

Le mot clef WHERE suivi d'une formule logique permet de poser une condition sur les lignes qui sont présentes dans le résultat. Les opérateurs de comparaisons des entiers (=, <, >, >=, <= et <> (différent de python !=). On peut utiliser aussi AND, OR et NOT pour combiner plusieurs condition.

Remarque : En SQL, contrairement au langage Python, on peut revenir à la ligne pour améliorer la lisibilité du programme. Pour la lisibilité, il est conseillé de le faire lorsque l'on doit écrire un mot clef. On évite les requêtes à rallonge sur une seule ligne

3) Écrire une requête SQL qui permet d'afficher le nom et le prénom des auteurs nés après 1970 compris et 1980 non compris.
Écrire une requête SQL permettant d'afficher le prénom de l'ami d'identifiant 5. Combien de résultat doit afficher cette requête au maximum ?

```
SELECT prenom, nom FROM Auteurs
WHERE annee_naissance >= 1970 AND annee_naissance < 1980
```

```
SELECT prenom FROM Amis
WHERE id = 5
```

On attend au maximum 1 seule

Remarque : On peut voir à ces réponses que l'on peut poser des conditions sur des colonnes qui ne sont pas sélectionnées dans le SELECT.

```
SELECT DISTINCT titre FROM Livre
```

titre
Harry potter
Mangeclous
L'informatique en CPGE
La vie de forêt
Carnet

Le mot clef DISTINCT permet de supprimer les lignes en doublons dans le résultat.

R1 :
```
SELECT id, prenom FROM Auteur LIMIT 3
```

id	Prenom
1	J .K
2	Etienne
3	Sophie

Résultat de R1

R2 :
```
SELECT id, prenom FROM Auteur LIMIT 3 OFFSET 3
```

id	Prenom
4	Sarah
5	Marcel
6	Albert Cohen

Résultat de R2

Le mot clef LIMIT permet d'indiquer le nombre maximum de lignes attendues dans le résultat. Si plus de lignes peuvent être présentes, elles ne sont tous simplement pas affichées.

Le mot clef OFFSET permet de décaler la première ligne affichée dans le cas où un LIMIT est utilisé.

⚠Le mot clef OFFSET ne s'utilise pas sans le mot clef LIMIT.

```
SELECT * FROM Auteur
ORDER BY annee_naissance ASC
```

id	Prenom	nom	annee_naissance
5	Marcel	Proust	1871
6	Albert Cohen	Cohen	1895
1	J .K	Rowling	1965
2	Etienne	Cochard	1970
3	Sophie	Rainero	1976
4	Sarah	Loulendo	1983

ORDER BY permet de sélectionner un attribut pour trier le résultat de la réponse en fonction de la l'attribut donné. DESC permet de trier par ordre décroissant, ASC par ordre croissant. S'il n'y a pas ASC ou DESC, le tri est croissant. Il est possible de trier selon un attribut puis un autre en séparant selon des virgules :

```
SELECT * FROM Auteur ORDER BY annee_naissance DESC, id ASC
```

Dans la requête précédente, on trie par ordre décroissant selon l'année de naissance, et pour une année de naissance égale, on trie par identifiant croissant.

ORDER BY se place après WHERE et avant LIMIT s'ils sont utilisés.

4) Donner une requête SQL permettant d'identifier le prénom et le nom de l'auteur le plus jeune.

```
SELECT prenom, nom
FROM Auteurs
ORDER BY annee_naissance DESC
LIMIT 1
```

3.3 Requêtes ensemblistes

```
SELECT nom, annee_naissance FROM auteur
WHERE annee_naissance <= 1900
UNION
SELECT nom, annee_naissance FROM auteur
WHERE annee_naissance >= 1980
```

nom	annee_naissance
Loulendo	1983
Proust	1871
Cohen	1895

UNION permet de faire l'union entre deux tables. Le résultat d'une requête est une table, donc il est possible de faire l'union de deux requêtes.

Une exigence est que les deux tables possèdent les mêmes colonnes. Sinon, faire une union n'a pas de sens.

Requête 1 :
```
SELECT titre FROM livre WHERE Langue == "francais"
INTERSECT
SELECT titre FROM livre WHERE Langue == "anglais"
```

Requête 2 :
```
SELECT titre FROM livre WHERE Langue == "francais"
EXCEPT
SELECT titre FROM livre WHERE Langue == "anglais"
```

5) Essayer de deviner le résultat des requêtes précédentes? Traduire ces requêtes par une phrase en français décrivant leur sens.

Les titres de livres que je possède en anglais et en français pour la requête 1.

Les titres de livres que je possède en français mais pas en anglais pour la requête 2.

3.4 Requête de jointure

Les requêtes que nous effectuons pour l'instant ne profitent pas des liens que nous avons entre les tables. On va voir alors l'opérateur de jointure, qui permet grâce aux clefs étrangères de lier deux tables.

L'idée de la jointure est de prendre une table A qui possède une clef étrangère faisant référence à une clef primaire d'une table B, et pour tout n-uplet de la table A de clef étrangère id_B, on rajoute comme attribut les colonnes de la table B correspondant au n-uplet possédant id_B.

id	titre	id_editeur	Langue
1	Harry potter	1	français
2	Harry potter	3	anglais
3	Mangeclous	1	français
4	L'informatique en CPGE	2	français
5	La vie de forêt	4	français
6	Carnet	1	français

id	nom
1	Gallimard
2	Ellipse
3	Bloomsbury
4	Sens dessus Dessous

Exemple de jointure :

```
SELECT * FROM Livre
JOIN Editeur
ON Editeur.id = Livre.id_editeur
```

Livre.id	titre	id_editeur	Langue	Editeur.id	nom
1	Harry potter	1	français	1	Gallimard
2	Harry potter	3	anglais	3	Bloomsbury
3	Mangeclous	1	français	1	Gallimard
4	L'informatique en CPGE	2	français	2	Ellipse
5	La vie de forêt	4	français	4	Sens dessus Dessous
6	Carnet	1	français	1	Gallimard

Le nouveau mot clef est JOIN, il se met après FROM et nécessite d'utiliser un autre mot clef, ON. Après JOIN, on écrit le nom de la table que l'on veut joindre à la table après le FROM, et après ON, on écrit quel doit être l'identifiant en commun lors de la jointure (Pour que cela se passe bien ,il faut choisir la clef étrangère et la clef primaire).

Comme on commence à écrire avec deux tables différentes, il peut y avoir des confusions dans les noms d'attributs. On note alors à quel attribut on fait référence en notant les attribut Table.attribut. Ce n'est pas obligatoire si le nom de l'attribut n'est pas ambigu, c'est à dire qu'il n'existe que dans une des tables traitées. Si le nom existe dans les deux tables, cela est alors nécessaire.

⚠Si l'on doit joindre trois table, il ne faut pas commencer par joindre les deux tables qui n'ont pas des liens.

3.5 Agrégations

En SQL, on peut utiliser des fonctions prédéfinies, dans les attributs après le SELECT, pour effectuer des traitements sur les colonnes de notre table.

```
SELECT MIN(annee_naissance) FROM Auteurs
```

MIN(annee_naissance)
1871

Ce mot clef permet de sélectionner la valeur minimum de l'année de naissance. Si l'on affiche un attribut supplémentaire en dehors de la fonction d'agrégation, la valeur affichée sera celle de la ligne qui atteint le minimum.

```
SELECT nom, MIN(annee_naissance)
FROM Auteurs
```

nom	MIN(annee_naissance)
Proust	1871

6) Refaire la question 4) avec la fonction d'agrégation MAX.

```
SELECT COUNT(*) AS nb_auteur
FROM Auteurs
```

nb_auteur
6

La fonction d'agrégation COUNT permet de compter le nombre de ligne de notre base. Si l'on affiche un autre attribut à côté du compte, la valeur affichée sera choisi aléatoirement parmi les lignes comptées.

```
SELECT COUNT(*) AS nb_auteur
FROM Auteurs
WHERE annee_naissance > 1900
```

nb_auteur
4

Le mot clef WHERE s'applique avant d'effectuer la fonction d'agrégation. C'est à dire que la requête d'avant compte seulement les lignes où l'année de naissance de l'auteur est strictement supérieur à 1900.

```
SELECT AVG(annee_naissance) AS avg
FROM Auteurs
WHERE annee_naissance > 1900
```

avg
1973,5

La fonction d'agrégation AVG permet d'effectuer la moyenne de l'attribut donné en argument. Il existe aussi la fonction d'agrégation SUM, qui effectue la somme des attributs donnée.

7) Donner une requête SQL me donnant le nombre de livre qui n'est pas rendu.

```
SELECT Langue, COUNT(*) AS nb_livre
FROM Livre
GROUP BY Langue
```

langue	nb_livre
français	5
anglais	1

Le GROUP BY permet d'appliquer la fonction d'agrégation à plusieurs ensemble différents. La fonction COUNT va ici compter pour chaque valeur de l'attribut Langue le nombre de ligne correspondante.

A noter que le *A_ecrit* dans le ON est facultatif, mais pas *Livre* ou *Auteurs*, car le nom id est utilisé dans les deux tables.

3.6 Filtrage agrégats

On a vu que le WHERE, associé aux fonctions d'agrégations, permet de restreindre l'ensemble dans lequel s'applique une fonction d'agrégation.

Si l'on souhaite poser une condition sur le résultat d'une fonction d'agrégation, on va utiliser le mot clef HAVING

```
SELECT Langue, COUNT(*) AS nb_livre
FROM Livre
GROUP BY Langue
HAVING nb_livre > 1
```

langue	nb_livre
français	5

- WHERE : appliquer une condition avant les fonctions d'agrégations, restreindre les lignes où s'applique la fonction d'agrégation.
- GROUP BY : Permet de définir un attribut selon lequel on définit les ensembles sur lequel s'applique notre fonction d'agrégation.
- HAVING : Permet d'énoncer une condition sur le résultat de notre fonction d'agrégation.

8) Donner un requête SQL donnant le titre des livres et l'année de naissance moyenne des auteurs de ce livre, et n'afficher que ceux dont la moyenne est supérieur à 1950.

```
SELECT titre, AVG(annee_naissance)
FROM Livre
JOIN A_Ecrit ON Livre.id = A_ecrit.id_livre
JOIN Auteur ON Auteur.id = A_ecrit.id_auteur
GROUP BY id_livre
HAVING AVG(annee_naissance) >= 1950
```

3.7 Requêtes imbriquées

On a vu que la question 6 ne permet pas de trouver tous les auteurs les plus vieux. Pour trouver tous les auteurs les plus vieux, on peut utiliser une requête imbriquée. Le résultat d'une requête, s'il est réduit à une colonne et à une ligne, peut être utilisé comme un entier, flottant, chaîne de caractères.

```
SELECT prenom,nom
FROM Auteur
WHERE annee_naissance =
    (SELECT MAX(annee_naissance)
    FROM Auteurs)
```

3.8 Produit cartésien

Si l'on regarde les sections précédentes, on ne peut répondre qu'aux questions qui respectent les relations sur nos tables. Cependant, si l'on souhaite répondre à des questions qui crée de nouvelles relations, cela n'est pas possible.

On va donc voir le produit cartésien. Si l'on dispose de deux tables :

A : a1, a2
1 3
2 3
3 4

B : b1
x
z

Le produit cartésien de A et B donne :

A, B : a1, a2, b1
1 3 x
1 3 z
2 3 x
2 3 z
3 4 x
3 4 z

Pour chaque ligne de la table A, on la complète par chaque ligne de la table B. Donc, dans notre tables finale, si on a x lignes dans A et y lignes dans B, le produit cartésien A, B aura x*y lignes.

```
SELECT E1.id_amis AS id1, E2.id_amis as id2
FROM Emprunts AS E1, Emprunts AS E2
WHERE E1.id_livre = E2.id_livre AND
E1.id_amis <> E2.id_amis
```

id1	id2
1	3
3	1

La requête précédente peut se lire comme il suit :

- À chaque emprunt, j'associe un autre emprunt (Produit cartésien)
- Je ne garde que les lignes dont les emprunts concernent le même livre
- Je ne garde que les lignes dont les emprunts concernent deux amis différents

La requête précédente permet d'afficher des lignes composées de deux identifiants d'amis qui ont empruntés le même livre.

9) Comment faire pour que les couples de mêmes amis ne soient affichés qu'une fois.

- Ajout du mot clef DISTINCT
- E1.id_amis <> E2.id_amis devient E1.id_amis < E2.id_amis

4 Résumé d'une requête

Une requête s'écrit de cette manière, avec un ordre de mot clef qui doit être respecté :

```
SELECT attribut, fonction_agregation(attribut)
FROM table1, table2
JOIN table3 ON table1.clef_primaire = table3.clef_etrangere
WHERE condition1 AND/OR condition2
GROUP BY attribut
HAVING condition3
ORDER BY att1 ASC/DESC, att2 ASC/DESC
LIMIT int1 OFFSET int2
```

Ensuite, on peut utiliser plusieurs requêtes imbriquées. Une requête étant une table, on peut mettre après FROM le résultat d'une requête. On peut utiliser les opérateurs ensemblistes pour réunir le résultat de deux requêtes. Si une requête ne comporte qu'une valeur, on peut aussi l'utiliser dans un test d'égalité.

```
SELECT prenom FROM Auteurs
WHERE annee_naissance = (SELECT MAX(annee_naissance)
                         FROM Auteurs)
```

5 Exercice

10) Titres des livres dont la maison d'édition est Gallimard.

11) Titres des livres qui ne sont pas dans ma bibliothèque.

12) Noms et prénom de mes amis qui ont déjà empruntés un livre en anglais.

13) Titres des livres dont les auteurs sont nées après 1900 (trois jointures)

6 Exercice 1 (Mines-Pont 2023)

Une base de données stocke les informations liées aux polices de caractères dans 4 tables ou relations :

- Famille décrit les familles de polices, avec fid la clé primaire entière et fnom leur nom.
- Police décrit les polices de caractères disponibles, avec pid la clé primaire entière, pnom le nom de la police et fid de numéro de sa famille.
- Caractere décrit les caractères, avec code la clé primaire entière, car le caractère lui-même, cnom le nom du caractère.
- Glyphe décrit les glyphes disponibles, avec gid la clé primaire entière, code le code du caractère correspondant au glyphe, pid le numéro de la police à laquelle le glyphe appartient, groman un booléen vrai pour du roman et faux pour de l'italique et gdesc la description vectorielle du glyphe.

Famille

fid	fnom
1	Humane
2	Garalde
3	Réale
4	Didone
5	Mécane
6	Linéale
...	...

Police

pid	pnom	fid
1	Centaur	1
2	Garamond	2
3	Times New Roman	3
4	Computer Modern	4
...	...	...
21	Triangle	6
...	...	...

Caractere

code	car	cnom
65	A	lettre majuscule latine a
66	B	lettre majuscule latine b
...	...	...
97	a	lettre minuscule latine a
98	b	lettre minuscule latine b
99	c	lettre minuscule latine c
...	...	...

Glyphe

gid	code	pid	groman	gdesc
1	65	20	True	`[ [ [0, 0], [1, 2], [2, 0] ], [ [0.5, 1], [1.5, 1] ] ]`
2	65	20	False	`[ [ [0, 0], [2, 2], [2, 0] ], [ [1, 1], [2, 1] ] ]`
...	...	...	...	...
501	97	21	True	`[ [ [0, 0], [0.5, 1], [1, 0], [0, 0] ] ]`
502	98	21	True	`[ [ [0, 2], [0, 0], [1, 0.5], [0, 1] ] ]`
503	99	21	True	`[ [ [1, 1], [0, 0.5], [1, 0] ] ]`
504	100	21	True	`[ [ [1, 2], [1, 0], [0, 0.5], [1, 1] ] ]`
...	...	...	...	...

1) Proposer une requête en SQL sur cette base de données pour compter le nombre de glyphes en roman (cf. description précédente).

```
SELECT COUNT(*) FROM Glyphe
WHERE groman = True
```

2) Proposer une requête en SQL afin d'extraire la description vectorielle du caractère A dans la police nommée Helvetica en italique.

```
SELECT gdesc FROM Glyphe
JOIN Caractere ON Glyphe.code = Caractere.code
JOIN Police ON Police.pid = Glyphe.pid
WHERE car = "A" AND pnom = 'Helvetica'
AND groman = False
```

3) Proposer une requête en SQL pour extraire les noms des familles qui disposent de polices et leur nombre de polices, classés par ordre alphabétique.

```
SELECT fnom, COUNT(*) AS nb FROM Famille
JOIN Police ON Famille.fid = Police.fid
GROUP BY fid
HAVING nb > 0
ORDER BY fnom ASC
```

7 Exercice 2

Voici une base de données d'une société organisant des croisières et son schéma.

- Croisiere(cid : string, depart : string, arrivee : string, distance : integer)
- Bateau(bid : string, type : string, autonomie : integer)
- Navigable(bid : string, eid : string)
- Employe(eid : string, nom : string, age : integer)

La relation Employe contient tous les employés de la compagnie dont les marins pilotant les bateaux. La relation Navigable indique pour chaque bateau quels sont les marins pouvant piloter ce bateau. Dans la relation Navigable, l'attribut eid est une clef étrangère provenant de la relation Employe et l'attribut bid est une clef étrangère provenant de la relation Bateau. Un marin est un employé qui peut piloter au moins un bateau. Les clefs primaires d'une relation sont les attributs qui sont soulignés. Les distances et les autonomies sont exprimées en nœuds.

7.1 Questions

1) Écrire la requête suivante en SQL, donner les marins qui peuvent piloter des catamarans. Renvoyer les identifiants de ces marins.

```
SELECT DISTINCT eid FROM Navigable
JOIN Bateau ON Navigable.bid = Bateau.bid
WHERE type = "Catamaran"
```

2) Écrire la requête suivante en SQL, donner les identifiants des bateaux et les identifiants de croisières tels que les bateaux peuvent effectuer les croisières sans réapprovisionnement.

```
SELECT cid, bid
FROM Bateau, Croisiere
WHERE autonomie > distance
```

3) Écrire la requête suivante en SQL, donner l'identifiant des marins qui peuvent piloter des bateaux pouvant parcourir strictement plus de 3000 nœuds mais qui ne peuvent pas piloter des catamarans.

```
SELECT DISTINCT eid
FROM Navigable
JOIN Bateau ON Bateau.bid = Navigable.bid
WHERE autonomie > 3000
EXCEPT
SELECT DISTINCT eid
FROM Navigable
JOIN Bateau ON Bateau.bid = Navigable.bid
WHERE type = 'catamaran'
```

4) Écrire la requête suivante en SQL, donner l'identifiant des employés les plus âgés.

```
SELECT eid
FROM Employe
WHERE age = (SELECT max(age)
             FROM Employe)
```

5) Écrire la requête suivante en SQL : donner le nombre de bateaux dans la base de données.

```
SELECT COUNT(*)
FROM Bateau
```

6) Écrire la requête suivante en SQL : pour chaque bateau, renvoyer son identifiant et le nombre de marins qui peuvent le piloter.

```
SELECT bid, COUNT(*) as nb_marins
FROM Navigable
GROUP BY bid
```

7) Écrire la requête suivante en SQL : donner l'âge moyen des marins.

```
SELECT AVG(age) FROM
(SELECT DISTINCT eid FROM Navigable) AS Marin
JOIN Employe ON Marin.eid = Employe.eid
```

8) Écrire la requête suivante en SQL, donner l'identifiant des bateaux pouvant être pilotés par tous les marins d'un âge strictement supérieur à 40 ans.

```
SELECT COUNT(eid) FROM
    (SELECT DISTINCT eid FROM Navigable
    JOIN Employe ON Employe.eid = Navigable.eid
    WHERE age > 40)
```

Je nomme R1 la requête précédente qui me donne le nombre de marin de plus de 40 ans.

```
SELECT bid, COUNT(*) as nb FROM Navigable
JOIN Employe ON Employe.eid = Navigable.eid
WHERE age > 40
GROUP BY bid
HAVING nb = R1
```