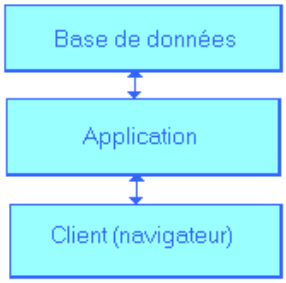
Bases de données - Langage SQL - TP n°1 - Enoncé

Objectif ✔ : concepts liés aux bases de données et langage SQL.
Savoir : vocabulaire associé aux bases de données, syntaxe des requêtes SQL.
Savoir faire : écriture de requêtes SQL.

1 Architecture trois tiers

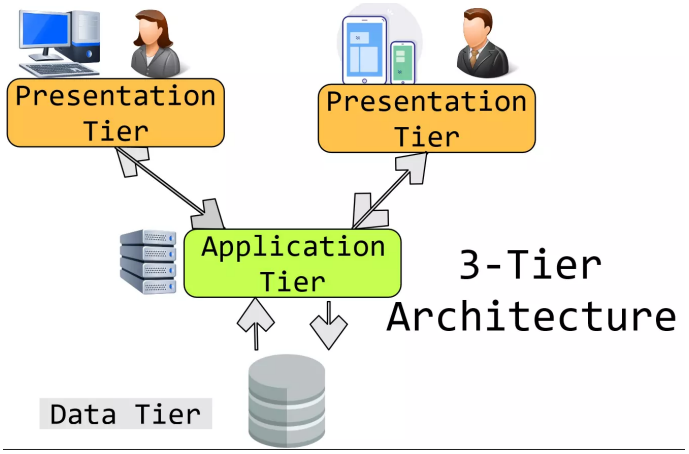
Ce nom provient de l'anglais *tier* signifiant étage ou niveau.
Il s'agit d'un modèle logique d'architecture applicative qui vise à modéliser une application comme un empilement de trois couches logicielles (étages, niveaux, tiers ou strates) dont le rôle est clairement défini.

- la présentation des données (affichage, dialogue avec l'utilisateur) ;
- le traitement des données ;
- l'accès aux données persistantes (données destinées à être conservées dans la durée, voire de manière définitive).



Ces trois niveaux sont, en général distribués sur trois machines :

- la présentation des données est réalisée sur l'interface utilisateur (navigateur internet, app smartphone, tablette, ordinateur...) ;
- le traitement des données est effectué par un programme exécuté sur un serveur distant ;
- les données elles-mêmes sont conservées dans une base de données sur un autre ordinateur distant (en général dupliquée sur un autre ordinateur distant pour des raisons de sécurité).



Cette répartition des rôles constitue un facteur de sécurité matérielle (panne matériel) et logicielle car elle permet d'intervenir séparemment sur les programmes (mises à jour) et les données (ajouts, suppressions...).
C'est aux bases de données qu'est consacrée la suite de ce TP

2 Introduction aux bases de données

2.1 La structuration des données

Afin d'introduire les concepts liés aux bases de données et ensuite tout au long de ce premier TP, nous utiliserons l'exemple d'une bibliothèque de prêt : gestion des livres, des emprunteurs, des emprunts...

2.1.1 Le tableur - Une très mauvaise solution

Exemple de tableur qui pourrait être utilisé pour gérer une bibliothèque.

index	titre	genre	auteurN	auteurP	nom	prénom	tél	date	retour
1	Salambo	Roman	Flaubert	Gustave	Castel	Claude	0612345112	12-août-10	23-août-10
2	93	Roman	Hugo	Victor	Le Dray	Camille	0422113827	17-sept.-10	03-oct.-10
3	Mme Bovary	Littérature	Flaubert	Gustave	Loutard	Annie	064331282	01-nov.-10	02-déc.-10
1	Salambô	Roman	Flaubert	Gustave	Filatre	Jean	0432168719	02-sept.-10	14-sept.-10
4	Aphorismes	Littérature	Wilde	Oscar	Castel	Marie	0412324494		
5	L'immoraliste	Roman	Gide	André	Biraud	Michèle	0434578612	05-févr.-11	
3	Madame Bovary	Roman	Flaubert	Gustave	Castel	Claude	0412324494	06-janv.-10	
2	Quatrevingt-treize	Roman	Hugo	Victor	Filâtre	Jean	0432168719	20-oct.-10	03-nov.-10
1	Salambo	Littérature	Flaubert	gustave	Ledray	Camille	0422113627	16-sept.-10	27-sept.-10
…	…	…	…	…	…	…	…	…	…

✎ Identifier dans ce tableau quelques causes de problèmes lors d'un traitement des données issues ce tableur.

Limites des structures de données plates (type feuille de calcul) pour le stockage et la recherche d'informations :

- **Intégrité / cohérence** des données (doublons orthographiques, formats variables ⇒ problèmes pour les recherches).
- **Efficacité des recherches** (lecture séquentielle de la totalité du tableau ⇒ complexité en temps).

2.1.2 Solution - Modèle relationnel

L'idée consiste à rassembler les données dans des structures plus petites, appelées **tables**, en fonction des **relations** que certaines données entretiennent entre elles.
Ainsi, il paraît judicieux de prévoir une table Auteurs contenant le nom, le prénom ainsi qu'un **numéro d'identification unique (id)** en cas d'homonymie.

Auteurs		
id	nom	prénom
1	Flaubert	Gustave
2	Hugo	Victor
3	Gide	André
4	Wilde	Oscar
…	…	…

Bien entendu, les données étant éclatées dans différentes tables, pour communiquer et effectuer des opérations (recherches, traitements) sur les bases de données, on utilise un **système de gestion de base de données** (abr. SGBD et en anglais DBMS pour database management system) : SQLite, MySQL (Oracle), PostgreSQL
Un **langage spécifique** nommé **SQL (Structured Query Language)** ou langage de requêtes structurées permet d'effectuer des requêtes (opérations) sur les bases de données.

Il existe des interfaces graphiques permettant d'effectuer ces requêtes dans un environnement graphique (permettant également de visualiser les tables et de les modifier) : phpMyAdmin, SQLiteStudio Cette fonction sera assurée cette année par un notebook muni du langage SQL.

2.1.3 Exemple - Bibliothèque de prêt

Extrait du fichier initial :

index	titre	genre	auteurN	auteurP	nom	prénom	tél	date	retour
1	Salambo	Roman	Flaubert	Gustave	Castel	Claude	0612345112	12-août-10	23-août-10

Le fichier initial est décomposé en 5 tables : Livres, Auteurs, Emprunts, Emprunteurs et Genre.

Livres

id	auteur	titre	genre
1	1	Salambô	1
2	2	Quatre-vingt-treize	1
3	3	L'immoraliste	1
4	4	Aphorismes	2
5	1	Madame Bovary	1
…	…	…	…

Auteurs

id	nom	prénom
1	Flaubert	Gustave
2	Hugo	Victor
3	Gide	André
4	Wilde	Oscar
…	…	…

Emprunts

qui	quoi	date	rendu
1	1	12-août-10	23-août-10
6	1	02-sept.-10	14-sept.-10
2	1	16-sept.-10	27-sept.-10
2	2	17-sept.-10	03-oct.-10
6	2	20-oct.-10	03-nov.-10
3	5	01-nov.-10	03-déc.-10
1	5	06-janv.-11	NULL
5	3	05-févr.-11	NULL
…	…	…	…

Emprunteurs

id	nom	prénom	tél
1	Castel	Claude	0612345112
2	Le Dray	Camille	0422113827
3	Loutard	Annie	0654331282
4	Castel	Marie	0412324494
5	Biraud	Michèle	0434578612
6	Filâtre	Jean	0432168719
…	…	…	…

Genre

id	genre
1	Roman
2	Littérature
…	…

La table Livres fait référence aux auteurs via un numéro d'identification qui renvoie à la table Auteurs. De même pour le genre.
Les autres relations sont analysées ci-dessous.

Aucune répétition : cohérence des données

Dans cet exemple, la feuille de calcul initiale a été remplacée par 5 **tables**, regroupant les données selon les **relations** qui les lient (renseignements concernant les emprunteurs, les auteurs, les livres... répartis dans autant de tables).
On voit apparaître dans ces tables des colonnes supplémentaires qui n'apparaissaient pas dans la feuile de calcul initiale : ces colonnes vont jouer un rôle essentiel pour **relier les tables** entre elles et reconstituer ainsi des liens entre différentes données.

A ce stade, il serait nécessaire de préciser comment relier les tables entre elles mais nous allons introduire parallèlement les concepts théoriques et les éléments de langage SQL, tout au long de ce TP.
Un résumé des requêtes SQL sera ensuite fourni.

2.2 Table - Attribut - Enregistrement ou n-uplet

Vocabulaire à connaître

Exemple	Table formelle	Définitions
Emprunteurs (id, nom, prénom, tél) — 1 Castel Claude 0612345112 ; 2 Le Dray Camille 0422113827 ; 3 Loutard Annie 0654331282 ; 4 Castel Marie 0412324494 ; 5 Biraud Michèle 0434578612 ; 6 Filâtre Jean 0432168719 ; …	Table (A_1, A_2, A_3, A_4) — n-uplet ou enregistrement	*Attributs* = noms des **colonnes**. *n-uplet* (tuple) ou *enregistrement* = une **ligne**.

2.3 Base de données

On utilise dans ce TP la base de données d'une bibliothèque (base pédagogique contenant très peu de données ; cf. TP suivant pour l'utilisation d'une base de données plus réaliste).
La base de données contient les tables "Auteurs", "Emprunteurs", "Emprunts", "Genre", "Livres" (cf. document donnant les tables et le graphe des clés).

🛠 **Pour comprendre l'élaboration des requêtes, ne pas hésiter à tester des requêtes partielles pour juger des effets des perfectionnements successifs.**

3 Affichage table ou résultat requête - SELECT ... FROM ...

Affichage de la totalité d'une table
SELECT * FROM *nom_table*

En informatique, le symbole * signifie généralement « tout » : la requête signifie donc « sélectionner toutes les lignes et toutes les colonnes de la table dont le nom est *nom_table* ».

Il est vivement recommandé de **commenter le code** et comme en python, il existe deux possibilités.

Commentaires
/* Ceci est commentaire au milieu du code ou sur plusieurs lignes */
-- Ceci est un commentaire sur une seule ligne
SELECT attribut / * On peut sélectionner un ou plusieurs attributs */ FROM nom_table

✏ Requête n°0
Afficher les enregistrements de la table *Emprunteurs*.

4 Filtrage - Tri - ORDER BY ... [DESC], LIMIT, OFFSET

Tri
/* Tri par ordre croissant */
SELECT * FROM *nom_table* ORDER BY *attribut*
/* Tri par ordre décroissant */
SELECT * FROM *nom_table* ORDER BY *attribut* DESC

Rappel : un **attribut** est le nom d'une colonne.
💡 Dans la syntaxe telle qu'on peut la trouver dans la documentation en ligne par exemple, ORDER BY [DESC], ***les crochets signifient que le mot clé est optionnel (si le mot clé est utilisé, il ne faut pas écrire les crochets)***.

✏ Requête n°1.a
Trier les enregistrements de la table *Emprunteurs* par ordre alphabétique sur le nom par ordre lexicographique croissant.

✏ Requête n°1.b
Modifier la requête pour obtenir un tri par ordre décroissant.

Filtrage des résultats de la requête
/* n résultats à partir du 1er */
SELECT * FROM *nom_table* LIMIT *n*
/* n résultats à partir du résultat m */
SELECT * FROM *nom_table* LIMIT *m, n*
/* n résultats à partir du résultat m : variante avec OFFSET */
SELECT * FROM *nom_table* LIMIT *mn* OFFSET *m*

LIMIT et OFFSET sont utiles avec les bases de données comportant de très nombreux enregistrements (cf. TP nř2).

Requête n°1.c
Choisir les 4 premiers résultats de la requête : "afficher la table *Emprunteurs*".

Requête n°1.d
Choisir les résultats 3 à 5 de la requête : "afficher la table *Emprunteurs*" sans utiliser OFFSET.

Requête n°1.e
Choisir les résultats 3 à 5 de la requête : "afficher la table *Emprunteurs*" en utilisant OFFSET.

Manipulation des données contenues dans une table.

Une requête doit permettre de sélectionner des informations pertinentes dans une colonne ou/et dans une ligne et non plus la table entière.
Sélectionner une **colonne** grâce à son **attribut** s'appelle une **projection**.
Sélectionner une **ligne** - un **enregistrement** - s'appelle une **sélection** ou **restriction**.

5 Projection

Projection - Sélection de colonne(s)
/* Il est possible de sélectionner une ou plusieurs colonnes en indiquant les attributs après SELECT */
SELECT [DISTINCT] *attribut1*, *attribut2* FROM *nom_table*

Requête n°2
Afficher les noms et les prénoms des emprunteurs (sans le mot clé [DISTINCT]).

Requête n°3
Quelle est la différence avec la requête suivante ?
SELECT DISTINCT *nom* FROM *Emprunteurs*

6 Sélection ou restriction - WHERE [NOT] - Formules de sélection

***Sélection* (d'enregistrements)**

SELECT * FROM *nom_table* WHERE *formule*

On appelle **formule de sélection** F une formule construite à partir des attributs, des fonctions usuelles, de constantes, des opérateurs de comparaison, de connecteurs logiques ...

Formule de sélection

formule peut être de la forme :

- *attribut* >, <, <= >=, =, <> ou != *nombre ou chaîne 'xxx'* [OR] [AND] ou toute opération mathématique (+ - * / ...) entre plusieurs attributs de la table
  Exemple : SELECT *attribut1* FROM *nom_table* WHERE *attribut1* + 2**attribut2* > 100 and *attribut1* < 10
- *attribut* LIKE '%ab%' ('%ab%' recherche la chaîne 'ab' à un endroit quelconque ; '%ab' à la fin ; 'ab%' au début)
- *attribut* BETWEEN *valeur1* AND *valeur2*
- *attribut* IS NULL (et non pas attribut=NULL)

⚠ Seules les **chaînes** doivent être entre guillemets. Les noms de tables et les attributs ne sont entourés de guillemets.

Requête n°4
Afficher les enregistrements de la table *Emprunteurs* de nom 'Castel' ou de prénom 'Jean'.

Requête n°5
Afficher les enregistrements de la table *Emprunteurs* ne s'appelant pas 'Castel'.

Requête n°6
Afficher les enregistrements de la table *Emprunteurs* dont le prénom contient un 'i'.

Requête n°7
Afficher les enregistrements de la table *Emprunteurs* dont l'*id* est compris entre 2 et 5.

Requête n°8
Afficher les enregistrements de la table *Emprunts* concernant les ouvrages non rendus.

6.1 Ecriture des requêtes

Dans les requêtes précédentes on a utilisé soit la projection soit la sélection/restriction cependant les deux sont en général utilisées simultanément.
Ce qui conduit à des requêtes de plus en plus longues.

🛠 Pour comprendre l'élaboration des requêtes, ne pas hésiter à tester des requêtes partielles pour juger des effets des perfectionnements successifs.

Pour faciliter la compréhension des requêtes longues, il est possible d'entrer les requêtes sous la forme :

```
SELECT ...
FROM ...
WHERE ...
```

Complément : deux requêtes successives doivent être séparées par ;

```
SELECT ... FROM ... ; SELECT ... FROM ...
```

7 Projection et sélection

✏ Requête n°9
Quels sont les noms des emprunteurs dont le prénom se termine par 'ie' ?

8 Agrégation - COUNT, SUM, MIN, MAX, AVG

Fonctions d'agrégation : COUNT, SUM, MAX, MIN, AVG (Average = moyenne).
Ces fonctions sont d'un emploi peu intuitif et seront introduites au fur et à mesure de la progression.
On peut les trouver derrière SELECT ou derrière HAVING (cf. paragraphe plus loin).

Fonction d'agrégation derrière SELECT

SELECT COUNT(*) FROM *nom_table* WHERE *formule*
Cette requête va compter le nombre d'enregistrements vérifiant la *formule* dans la table *nom_table*.
💡 Observer le résultat de la requête : COUNT(*) joue le rôle d'un **attribut** dans les noms de colonnes.

✏ Requête n°10.a
Afficher les enregistrements correspondant aux ouvrages non rendus.

✏ Requête n°10.b
Modifier la requête précédente de façon à compter le nombre douvrages non rendus.

✏ Requête n°10.c
💡 Une fonction d'agrégation peut être appliquée aux valeurs d'un attribut.
💻 👁 La requête ci-dessous n'a aucune signification mais elle illustre le fonctionnement d'une fonction d'agrégation appliquée à un attribut.

/* Requête 10.c - Agrégation */ SELECT AVG(qui) FROM Emprunts
✏ Requête n°10.d
La fonction MIN() (resp. MAX()) appliquée à des chaînes renvoie la chaîne dont la 1ère lettre est la plus proche de 'a' ou 'A' (resp. 'z' ou 'Z') dans l'ordre alphabétique.
Écrire la requête affichant le nom et le prénom de l'auteur possédant le dernier prénom dans l'ordre alphabétique.

✏ Requête n°10.e
La fonction LENGTH() appliquée à une chaîne renvoie sa longueur.
Écrire la requête affichant la longueur moyenne des titres.

9 Renommage - AS

Il est possible de renommer un attribut pour le rendre plus explicite.

Renommage

SELECT *attribut* AS *nouvel_attribut* FROM ...

⚠ *nouvel_attribut* sans guillemets et ne pas utiliser d'espace dans le nom (remplacer par un tiret _) ; si on utilise des guillemets, le comportement n'est pas tout à fait le même dans la suite.
✏ Requête n°11
Modifier la requête précédente pour que l'attribut du nombre d'ouvrages non rendus soit : Non_rendus.

10 Clés - Clé primaire - Clé étrangère

Les requêtes précédentes ne concernaient qu'une seule table, il reste à relier les tables entre elles.
Sur les tables ci-dessous, certains attributs font clairement référence à une autre table.
Par exemple, l'attribut *auteur* de la table *Livres* fait référence à l'attribut *id* de la table *Auteurs* :

Livres			
id	auteur	titre	genre
1	1	Salambô	1
2	2	Quatre-vingt-treize	1
3	3	L'immoraliste	1
4	4	Aphorismes	2
5	1	Madame Bovary	1
…	…	…	…

Auteurs		
id	nom	prénom
1	Flaubert	Gustave
2	Hugo	Victor
3	Gide	André
4	Wilde	Oscar
…	…	…

Emprunts			
qui	quoi	date	rendu
1	1	12-août-10	23-août-10
6	1	02-sept.-10	14-sept.-10
2	1	16-sept.-10	27-sept.-10
2	2	17-sept.-10	03-oct.-10
6	2	20-oct.-10	03-nov.-10
3	5	01-nov.-10	03-déc.-10
1	5	06-janv.-11	NULL
5	3	05-févr.-11	NULL
…	…	…	…

Emprunteurs			
id	nom	prénom	tél
1	Castel	Claude	0612345112
2	Le Dray	Camille	0422113827
3	Loutard	Annie	0654331282
4	Castel	Marie	0412324494
5	Biraud	Michèle	0434578612
6	Filâtre	Jean	0432168719
…	…	…	…

Genre	
id	genre
1	Roman
2	Littérature
…	…

La table Livres fait référence aux auteurs via un numéro d'identification qui renvoie à la table Auteurs. De même pour le genre.

De même, l'attribut *genre* de la même table *Livres* fait référence à l'attribut *id* de la table *Genre*.
La table *Livres* est donc reliée à deux autres tables, *Auteurs* et *Genre*.
Ces relations entre tables sont réalisées à partir d'attributs qui n'apparaissaient pas dans le tableur initial, ces attributs suppélementaires sont appelés des **clés**.
On distingue deux types de clés : les **clés primaires** et les **clés étrangères**.

Dans les 4 tables *Livres*, *Auteurs*, *Emprunteurs* et *Genre* (mais pas dans la table *Emprunts*) apparaît un attribut noté **id** permettant d'**identifier un et un seul enregistrement** de la table.
Cet attribut est la **clé primaire** qui permet une indexation des données (comme l'index d'une liste), ce qui rend les procédures d'interrogation de la table (requêtes) plus efficaces.
Dans la table *Emprunts*, la clé primaire est constituée des 3 attributs *qui*, *quoi*, *date* (il faut au moins 3 attributs pour identifier de façon unique un enregistrement dans cette table).

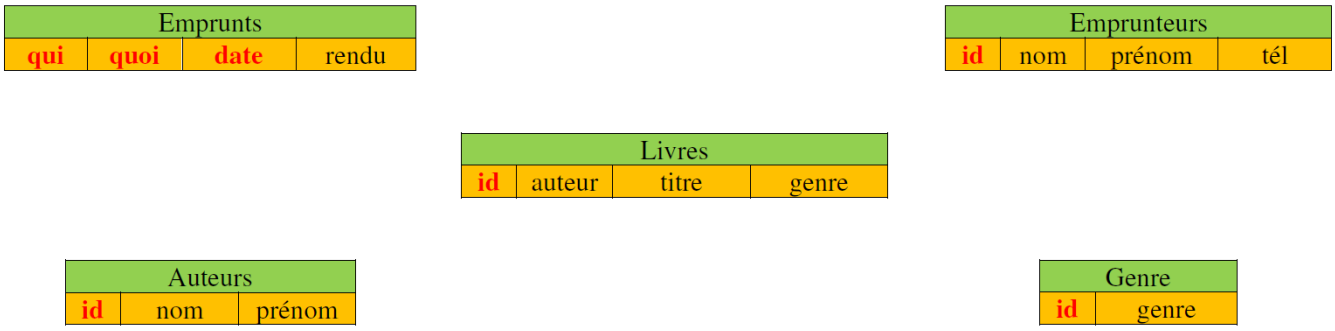
Clé primaire (Primary Key)

- Une **clé** (pour une relation, i.e. pour une table) est un **ensemble d'attributs** qui donne accès à un **unique enregistrement**.
- Une **clé primaire** est une clé de **taille minimale** (souvent notée id ou PK pour primary key, une clé primaire est en général soulignée dans les tables et les diagrammes fournis dans la documentation des bases de données).
- Une **clé secondaire** est une clé autre que la clé primaire.
- Dans une requête, pour faire référence à une clé (quelle q'elle soit) dans une table, on utilise la syntaxe ***table.cle*** dans laquelle la table figure avant le point et la clé après le point.

La table *Emprunts* possède un attribut *quoi* (noté *Emprunts.quoi*) qui fait le lien avec l'attribut id de la table *Livres* (noté *Livres.id*).
Cet attribut *quoi* constitue une **clé étrangère** de la table *Emprunts* : *quoi* fait référence à une clé primaire *id* d'une autre table, *Livres*.
De même l'attribut *Emprunts.qui* est une clé étrangère faisant le lien avec l'attribut *Emprunteurs.id*.

Clé étrangère (Foreign Key)
Une **clé étrangère** fait référence à une clé primaire d'une autre table.

Le **graphe des clés étrangères** est fourni par les concepteurs des bases de données (cf. TP n°2) mais, dans le cadre d'exercices sur des bases simples comme c'est le cas ici, il peut facilement être déduit des noms des attributs.
Compléter le diagramme suivant (graphe des clés étrangères) par des flèches associant les clés étrangères de chaque table aux clés primaires adéquates (faire en sorte que les flèches ne se croisent pas).
Indiquer au-dessus de chaque flèche, une légende de la forme ***table 1*** [***attribut***] ↔ ***table 2*** [***attribut***].



11 Jointure - JOIN ... ON ...

Pour relier deux tables entre elles on utilise la syntaxe suivante (et le graphe des clés étrangères).

Jointure
SELECT * FROM *table1* JOIN *table2* ON *table1.cle1* = *table2.cle2*
⚠ L'une des deux clés doit être une clé primaire, l'autre une clé étrangère ; la position de part et d'autre du signe « = » est sans importance.

Requête n°12.a
Réaliser la jointure entre les tables *Emprunteurs* et *Emprunts* avec les clés correctes et observer le résultat de la requête.

Il est possible d'utiliser des alias afin d'alléger la syntaxe des requêtes.

Utilisation d'alias
SELECT * FROM *table1* AS *t1* JOIN *table2* AS *t2* ON *table1.cle1* = *table2.cle2* *t1* est l'alias de *table1* et *t2* est l'alias de *table2*.
Il est possible d'effectuer des jointures multiples :
t1 JOIN *t2* ON *t1.cle1*=*t2.cle2* JOIN *t3* ON *t2.cle2*=*t3.cle3*.
L'ordre de la jointure est indifférent.

Requête n°12.b
Réaliser la jointure entre les tables *Livres* et *Auteurs* en utilisant les alias *L* et *A*.

Il est à présent possible de réaliser des requêtes plus intéressantes en combinant projection, restriction et jointure.

Requête n°13
Afficher les noms des emprunteurs ainsi que la date des emprunts.

Requête n°14
Afficher les noms et prénoms des emprunteurs des ouvrages non rendus.

Requête n°15
Combien de fois « Salambô » a-t-il été emprunté ?

12 Jointures multiples, sélections, projections

✏ Requête n°16
Quels sont les livres empruntés par Claude Castel ?

✏ Requête n°17
Qui a emprunté « Salambô » ?

13 Regroupements d'enregistrements - GROUP BY

💻 👁 Lors d'une jointure, certaines valeurs apparaissent plusieurs fois (comme dans le tableur de l'exemple initial).

```
/* Exécuter la requête et observer les colonnes nom et prénom */

SELECT *
FROM Emprunts JOIN Emprunteurs ON Emprunts.qui=Emprunteurs.id
```

On souhaite dans la suite, regrouper les enregistrements possédant des valeurs identiques pour certains attibuts (nom ici) en vue de faire des calculs.

📚 ***GROUP BY***
SELECT *attribut1*, *attribut2*, ... FROM *table* GROUP BY *attribut*
La clause GROUP BY permet de **regrouper** les n-uplets selon un critère.

```
/* Regroupement : exécuter la requête et comparer au résultat précédent */
SELECT nom,prénom
FROM Emprunts JOIN Emprunteurs ON Emprunts.qui=Emprunteurs.id
GROUP BY nom
```

💡 La table générée comporte moins de lignes : les lignes ayant le même attribut *nom* sont regroupées mais ces regroupements ne sont pas visibles sur la sortie écran.
Ces regroupements permettent ensuite d'effectuer des calculs.

💻 👁 Afficher le nombre d'emprunts de chaque emprunteur : la fonction d'agrégation COUNT(*) est ajoutée dans les attributs.

```
/* Regroupement et calculs : exécuter la requête et observer le résultat */
SELECT nom, COUNT(*)
FROM Emprunts JOIN Emprunteurs ON Emprunts.qui=Emprunteurs.id
GROUP BY nom
```

💡 La commande COUNT(*) permet de compter le nombre d'enregistrements par regroupement.

14 Sélection à l'intérieur d'un regroupement - GROUP BY...HAVING

📚 Il est ensuite possible d'effectuer une **sélection** parmi les regroupements grâce à une clause HAVING située **après** une clause GROUP BY.

📚 ***GROUP BY ... HAVING ...***
SELECT *attribut1*, *attribut2*, ... FROM *table* GROUP BY *attribut* HAVING *formule*

📚 ***Fonction d'agrégation derrière HAVING***
Dans la syntaxe
HAVING *formule*
formule **peut contenir une fonction d'agrégation.**

💻 👁 Requête n°18
Qui a emprunté au moins deux livres ?

```
/* Requête 18 - Regroupement et sélection dans le regroupement : exécuter la requête et observer le
    résultat */

SELECT nom, COUNT(*)
FROM Emprunts JOIN Emprunteurs ON Emprunts.qui=Emprunteurs.id
GROUP BY nom
HAVING COUNT(*) > 1
```

Il est possible d'utiliser le renommage afin d'avoir un affichage plus explicite.

```
/* Amélioration de l'affichage : ne pas utiliser le renommage derrière HAVING */

SELECT nom, COUNT(*) AS Nombre\_emprunts
FROM Emprunts JOIN Emprunteurs ON Emprunts.qui=Emprunteurs.id
GROUP BY nom
HAVING Nombre\_emprunts > 1
```

Remarque - COUNT(*) n'est pas indispensable dans la liste des attributs si on désire uniquement les noms :

```
/* Remarque */
SELECT nom
FROM Emprunts JOIN Emprunteurs ON Emprunts.qui=Emprunteurs.id
GROUP BY nom
HAVING COUNT(*) > 1
```

La clause GROUP BY **regroupe** les n-uplets par *nom* et la clause HAVING effectue une **sélection** à l'intérieur de ce regroupement en utilisant une fonction d'agrégation.

⚠ A ce stade, seule la fonction d'agrégation COUNT(*) a été utilisée car les fonctions d'agrégation MIN, MAX, SUM et AVG sont plus difficiles à mettre en oeuvre : elles nécessitent l'utilisation de sous-requêtes (à suivre...).

Ecriture des requêtes

Utiliser le résumé "Langage SQL" pour rédiger les requêtes et "Annexe TP" pour la description de la base de données utilisée.

15 Regroupements d'enregistrements et sélection, agrégation

Requête n°19
Afficher le nombre d'emprunts de chaque titre (à l'aide d'une seule requête). Utiliser un renommage.

Requête n°20
Quel livre a été emprunté trois fois ? Trier par ordre alphabétique.

On envisage dans la suite des requêtes plus complexes, imbriquées les unes dans les autres.

Une **sous-requête** permet de **générer une nouvelle table** dans laquelle la **requête principale** va extraire des informations.
La sous requête peut être placée soit derrière la clause WHERE soit derrière la clause FROM.

16 Sous-requêtes derrière WHERE

Sous-requête derrière ***WHERE***
SELECT ... FROM ... WHERE *formule* (SELECT ... FROM ... WHERE ...)
Où *formule* peut être de la forme :

- *attribut* >, <, <= >=, =, <> ou != *nombre ou chaîne 'xxx'* [OR] [AND] ou toute opération mathématique (+ - * / ...) entre plusieurs attributs de la table
- *attribut* [NOT] IN
- *attribut* [NOT] EXISTS
- *attribut* =, >, <, <>, >=, <= suivi de ALL (vrai pour toute entrée)
- *attribut* = >, <, <>, >=, <= suivi de ANY ou SOME (vrai pour au moins une entrée)

Requête n°21
Quels livres n'ont jamais été empruntés ?
→Commencer par écrire la sous-requête correspondant à la question : "Quels sont les titres empruntés au moins une fois ? ".
→Utiliser la clause `NOT IN` dans la formule de la requête principale.

Perfectionner la requête pour afficher également le nom de l'auteur.

17 Sous-requêtes derrière FROM

Pour utiliser les fonctions d'agrégation MIN, MAX, SUM ou AVG il faut parfois commencer par générer une table, en utilisant un regroupement, afin de générer des nombres correspondants aux enregistrements regroupés.

Sous-requête derrière ***FROM***
SELECT ... FROM (SELECT ... FROM ... WHERE ...) WHERE ...

Pour répondre à la question « Quel est le livre le plus emprunté ? », il faut commencer par générer la table dérivée contenant le nombre d'emprunts par titre (cf. ci-dessus requête n°19) :
`SELECT  AS NbreEmprunts FROM  GROUP BY` (sous-requête)

On cherche ensuite le maximum de la colonne correspondant au nombre d'emprunts :
`SELECT MAX(NbreEmprunts), ... FROM (SELECT  AS NbreEmprunts FROM  GROUP BY )` (requête complète)

Requête n°22
Quel est le livre le plus emprunté ?

18 Exercices

Dans le moteur SQL de ce notebook, les dates sont des chaînes de la forme : '2010-08-12' (année-mois-jour).

Vérifier en exécutant la requête :
`SELECT DATE FROM Emprunts`

Remarque : il existe des fonctions liées au dates en SQL mais elles ne sont pas implémentées dans ce notebook (et elles ne sont pas au programme).

Ex n°1
Combien y a-t-il eu d'emprunts en 2010 ?

Ex n°2
Combien de livres ont-ils été empruntés en septembre (quelle que soit l'année) ?

Ex n°3
Combien de livres ont été empruntés en septembre mais n'ont pas été rendus en septembre ?

19 Exercices (facultatifs)

✏ Requête n°23
Quels sont les livres empruntés par les lecteurs ayant emprunté au moins deux livres (trier par lecteur) ?

On constate que le code est redondant.
Il existe une solution : la création d'une **vue** (table "virtuelle").
`CREATE VIEW` *`nom_vue`* `AS` *`requête`*`;`
`SELECT  FROM table` *`nom_vue`*

Remarque : `DROP VIEW` *`nom_vue`* pour suprimer une vue.

💻 👁 Requête n°23 - Avec vue
Quels sont les livres empruntés par les lecteurs ayant emprunté au moins deux livres (trier par lecteur) ?

```
 /* Requête 23 avec vue : 1ère étape = création de la vue*/
-- Création de la vue
CREATE VIEW tableComplete AS
SELECT *
FROM (Emprunts JOIN Livres ON Emprunts.quoi=Livres.id) JOIN Emprunteurs ON Emprunts.qui=Emprunteurs.id;

-- Affichage de la vue
SELECT * FROM tableComplete
```

```
 /* Requête 23 avec vue : 2ème étape = utilisation de la vue */
SELECT nom, titre FROM tableComplete
WHERE nom IN
(SELECT nom FROM tableComplete GROUP BY nom HAVING COUNT(*)>1)
ORDER BY nom
```

✏ Requête n°24
Qui n'a emprunté que du Flaubert ?

✏ Requête n°25
Qui a emprunté au moins 1,2 fois plus de livres que la moyenne ? (Avec vue)

✏ Requête n°26
Quels livres ont été empruntés à la fois par Le Dray et Filâtre ? (Avec vue)