Extrait du programme Information et Gestion
pour les 2 spécialités de Première Sciences et Technologies de la Gestion

2.3- La base de données

THÈMES	SENS ET PORTÉE DE L' ÉTUDE	NOTIONS ET CONTENUS À CONSTRUIRE
2.3. La base de données *(10 heures)*	La base de données est un ensemble d'informations structurées en tables dont l'implantation, la mise à jour et l'exploitation sont réalisées à l'aide d'un système de gestion de bases de données relationnel. Elle est définie par son schéma (la structure) et son contenu (les valeurs).	- *Le schéma de la base de données* Table, enregistrement (ou ligne), champ (ou colonne) Prise en charge des contraintes d'intégrité : type, clé primaire et clé étrangère - *Le langage de requête SQL* Restriction, projection, jointure, tri - *La gestion de la base de données* La gestion des données : stockage, intégrité, sécurité Les droits d'accès aux données

III. INDICATIONS COMPLÉMENTAIRES ET LIMITES

2.3 La base de données

L'étude de l'organisation des données dans les tables (nom, type,clés) met en évidence les mécanismes utilisés par les SGBD pour assurer le contrôle de la cohérence des données lors d'une mise à jour.

Le recours au langage SQL fonde l'usage d'un raisonnement partant de l'expression du résultat à obtenir. Son apprentissage, indépendant de l'interface propre à chaque outil, contribue à l'acquisition progressive de la capacité de passer de l'expression d'un besoin en langage naturel à sa définition dans un langage formel.

L'étude est volontairement limitée à l'ordre SELECT en utilisant les clauses d'identification des sources de données sur une ou plusieurs tables (FROM), de restriction et de jointure (WHERE) et de tri (ORDER BY). Les fonctions usuelles d'agrégat et de traitement des chaînes de caractères, des nombres et des dates sont abordées en fonction des besoins. L'utilisation d'un « requêteur » graphique, pour autant qu'elle permette de mettre en œuvre une logique de construction structurée,peut faciliter la visualisation et la compréhension des requêtes exprimées en langage SQL.

SOMMAIRE

Rque : Le point I – B est hors programme STG. Il est présenté à titre d'information.

SQL

L'algèbre relationnelle est la base du développement du langage SQL (Structured Query Langage). Ce langage textuel permet de communiquer avec une base de données relationnelle. Il a été réalisé par IBM puis est devenu un standard ANSI (American National Standards Institute) approuvé par l'ISO (International Standards Organization) en 1987. SQL/92 est la norme très répandue. Le dernier standard validé est SQL3 appelé bien souvent SQL/99. Toutefois, pour avoir un langage plus complet, les concepteurs de SGBD/R ajoutent certaines fonctionnalités au SQL standard.

Parmi les différentes fonctionnalités SQL, il est possible de retrouver les catégories de commandes suivantes :
- DDL (Data Définition Language) qui permet de définir et de modifier le schéma d'une base de données relationnelle.
- DML (Data Manipulation Language) qui permet la modification d'une base de données relationnelle.
- DQL (Data Query Language) qui permet l'interrogation de la base de données relationnelle.
- DCL (Data Contrôle Language) pour contrôler la sécurité et l'intégrité de la base de données relationnelle.

Seules l'interrogation et la manipulation des données seront abordées.

I- Les requêtes SQL

A) Les requêtes simples

Toutes les requêtes réalisées à l'aide d'un requêteur graphique (QBE) peuvent être réalisées à l'aide d'ordres SQL et donc être directement utilisables sur ordinateur. Les ordres sont non procéduraux c.a.d que pour obtenir un résultat, il suffit d'indiquer ce que l'on désire obtenir. Il n'est pas nécessaire d'indiquer la manière d'y parvenir comme dans un langage procédural.

1- les principaux mots

Mots	Arguments
SELECT	Attribut(s)
FROM	Table(s)
WHERE	Condition(s) sur une ligne
GROUP BY	Attribut(s) de partitionnement
HAVING	Condition(s) de sélection sur un groupe de lignes
ORDER BY	Attribut(s) de tri

Les mots clés ne sont pas sensibles à la casse contrairement aux données.

Sous Access, il est possible de réaliser des requêtes SQL.
Pour cela, il est possible d'utiliser la méthode suivante :
- sélectionner l'objet Requête,
- fermer la fenêtre proposant d'ajouter les tables

Dans les menus apparaît avec en haut à gauche l'outil SQL. Il reste à taper le code

2- Les opérations

a) La projection

Rappel : Une projection permet d'extraire des colonnes spécifiées d'une table.
Une projection s'exprime à l'aide de SQL par la commande :

```
SELECT [ALL | DISTINCT] [colonne1 , colonne2 .....]
FROM  nomtable ;
```

Par défaut, SQL n'élimine pas les doubles à moins que cela soit explicitement demandé par le mot clé DISTINCT

Exemple : Pour obtenir la liste des noms des candidats avec leur date de naissance de la table CANDIDAT il sera nécessaire d'écrire :

```
SELECT NomCandidat, DateNaissance
FROM candidat ;
```

Si l'on souhaite éliminer les doubles, il faudrait noter :

```
SELECT DISTINCT NomCandidat, DateNaissance
FROM candidat ;
```

b) La sélection

On parle également à ce niveau de restriction.
Une sélection s'exprime à l'aide de SQL par la commande :

```
SELECT *
FROM  nomtable
WHERE  condition ;
```

Le symbole * signifie que l'on affiche toutes les colonnes d'une table. La combinaison avec une projection s'effectue en remplaçant * par la liste des colonnes à projeter.
Exemple 1 : On désire connaître les caractéristiques du ou des candidats dont le nom est FAURE.

```
SELECT *
FROM CANDIDAT
WHERE  NomCandidat IN('Faure') ;
```

La même requête avec l'affichage du NumCandidat et de la date de naissance s'écrit :

```
SELECT NumCandidat, DateNaissance
FROM CANDIDAT
WHERE  NomCandidat  IN('Faure') ;
```

De manière plus générale, la condition suivant la commande WHERE peut inclure différents opérateurs de comparaison et différents opérateurs booléens :

*** les opérateurs de comparaison**

SYMBOLE	SIGNIFICATION
=	Égal
<> =!	Différent
<	Inférieur
>	Supérieur
<=	Inférieur ou égal
>=	Supérieur ou égal

Avec l'opérateur Like, il est possible d'utiliser des caractères "joker" comme % et _ . Sous Access, le % est remplacé par * et _ par ?.
Le "joker" % remplace de 0 à n caractères quelconques. Exemple : Si l'on écrit après la commande WHERE attribut Like'F%', la condition portera sur toutes les valeurs de l'attribut dont la première lettre commence par F.
Le "joker" _ remplace un caractère quelconque et un seul. Exemple : Si l'on écrit après la commande WHERE attribut Like'F_ _ , la condition portera sur toutes les valeurs de l'attribut dont la première lettre commence par F et ayant ensuite seulement 2 autres caractères.

*** les opérateurs logiques**

SYMBOLE	SIGNIFICATION
Between...AND...	Entre ... et ...
Like	Comme
IS NULL	Dont la valeur est nulle
IN	Compris dans la liste

*** les opérateurs booléens**

SYMBOLE	SIGNIFICATION
AND	ET
OR	OU
NOT(1)	NON

(1) L'opérateur NOT inverse la valeur du résultat. Il est ainsi possible d'écrire : NOT BETWEEN, NOT IN, NOT LIKE, IS NOT NULL,

À ces opérateurs booléens, il peut être ajouté des parenthèses pour indiquer l'ordre d'évaluation.

***le tri**

Il est possible de trier les résultats suivant l'ordre ascendant (mot clé : ASC) ou descendant (mot clé : DESC) d'une ou plusieurs colonnes.
La commande SQL sera la suivante :
ORDER BY {expression | position} [ASC | DESC] [,{expression | position} [ASC | DESC] ....]

Exemple : on désire relever les candidats dont le nom est FAURE en affichant N°candidat et le lieu de naissance. Les données seront présentées dans l'ordre croissant des lieux de naissance. L'instruction SQL sera la suivante :

```
SELECT NumCandidat, Lieu  naissance
FROM CANDIDAT
WHERE  NomCandidat IN('Faure')
ORDER BY Lieu naissance ;
```

Si l'on veut obtenir les mêmes données mais en affichant les lieux de naissance dans un ordre décroissant, l'instruction SQL peut être la suivante :

```
SELECT NumCandidat, Lieu  naissance
FROM CANDIDAT
WHERE  NomCandidat IN('Faure')
ORDER BY 2 DESC ;
```

On remarque que l'on a utilisé une syntaxe qui consiste à indiquer la position de la colonne dans la commande SELECT.

c) la jointure

Cette opération consiste à joindre 2 tables avec un critère de sélection. Une sélection s'exprime à l'aide de SQL par la commande :

```
SELECT *
FROM  nomtable1, nomtable2
WHERE  nomtable1.clé=nomtable2.clé ;
```

Exemple : on veut joindre la table EMPRUNTER et la table CANDIDAT.
L'instruction SQL sera la suivante :

```
SELECT *
FROM  Candidat , Emprunter
WHERE  Candidat.NumCandidat=Emprunter.NumCandidat ;
```

Il est possible de renommer les tables par des alias. Cela permet notamment d'écrire plus rapidement le code pour réaliser des requêtes et de réduire les erreurs de frappe. Ainsi pour l'exemple précédent, il est possible d'écrire la même requête de la manière suivante :

```
SELECT *
FROM  Candidat C , Emprunter  E
WHERE  C.NumCandidat=E.NumCandidat ;
```

Avec le langage SQL, il est possible de combiner les différentes opérations.
Il est ainsi possible de donner la liste des livres (nom des livres, genre et code éditeur) empruntés par les candidats FAURE. Dans la table on fera également apparaître le N° candidat pour distinguer éventuellement les candidats ayant le même nom.
Les différentes instructions sont les suivantes :

```
SELECT C.NumCandidat, Nom livre, Genre, Code_Editeur
FROM Candidat C, Emprunter E, Livre L
WHERE NomCandidat IN('FAURE')
AND C.NumCandidat = E.NumCandidat
AND E.code_livre = L.code_livre ;
```

d) Les fonctions de calcul et les regroupements

*** Les fonctions de calcul**

Il est possible d'effectuer dans une requête SQL, des calculs horizontaux sur des lignes en utilisant les opérateurs +, -, *, / aussi bien dans la commande SELECT que dans la commande WHERE. Les arguments des opérateurs sont des noms de colonnes de type numérique ou des constantes.

Exemple : on souhaite afficher la table EMPRUNTER avec des dates d'emprunt décalées de 2 jours et ne pas afficher les dates de retour. L'instruction SQL sera la suivante :

```
SELECT NumCandidat, CodeLivre, DateEmprunt+2
FROM  Emprunter ;
```

Exemple : on souhaite afficher le numéro des candidats qui ont rendu le livre après 20 jours d'emprunt.

```
SELECT NumCandidat
FROM  Emprunter
WHERE DateRetour > DateEmprunt +20 ;
```

*** Les fonctions agrégatives**

Elles permettent d'effectuer des calculs verticaux pour l'ensemble ou un sous-ensemble des valeurs d'une colonne.
Les fonctions principales sont les suivantes :

FONCTIONS	SYMBOLE
SUM	permet d'effectuer la somme des valeurs d'une colonne numérique
AVG	permet d'effectuer la moyenne des valeurs d'une colonne numérique
MAX	permet de rechercher la valeur maximale d'une colonne numérique
MIN	permet de rechercher la valeur minimale d'une colonne numérique
COUNT(1)	permet de compter le nombre de valeurs d'une colonne

(1) - Pour compter le nombre de lignes sélectionnées, la fonction doit être utilisée avec l'argument *.
- Pour compter le nombre de valeurs distinctes prises par une colonne, il faut indiquer l'argument DISTINCT suivi de l'argument considéré.

- instructions portant sur l'ensemble d'une colonne

Il n'y a pas ici de difficultés particulières

Exemple 1
Vous souhaitez déterminer le nombre de candidats.

```
SELECT COUNT(NumCandidat) AS NbCandidat
FROM CANDIDAT ;
```

Exemple 2
Vous cherchez à déterminer le total des pages à lire.

```
SELECT SUM(NbrePages) AS SommePages
FROM LIVRE
```

Exemple 3
On désire déterminer la moyenne des pages pour les livres. La requête sera la suivante :

```
SELECT AVG(NbrePages) AS MoyPage
FROM LIVRE ;
```

- Instructions portant sur les sous-ensembles d'une colonne

° Le partitionnement

Il doit permettre d'effectuer des calculs statistiques pour chaque sous-ensemble de lignes vérifiant un même critère. Le partitionnement s'exprime en SQL par la commande GROUP BY suivie du nom des colonnes de partitionnement.
Exemple 1
Vous souhaitez déterminer le nombre d'emprunt pour chaque candidat ayant emprunté un livre.

```
SELECT NumCandidat, COUNT(NumCandidat)
FROM EMPRUNTER
GROUP BY NumCandidat ;
```

Exemple 2
Vous cherchez à déterminer le total des pages par code éditeur

```
SELECT CodeEditeur, SUM(NbrePages)
FROM LIVRE
GROUP BY CodeEditeur ;
```

Exemple 3
Vous cherchez à déterminer la moyenne du nombre moyen de pages par éditeur.

```
SELECT CodeEditeur, AVG(NbrePages)
FROM LIVRE
GROUP BY CodeEditeur ;
```

° Les conditions sur des classes de lignes.

Lorsqu'une condition de recherche doit être exprimée non pas sur les lignes (WHERE) d'une table mais sur les **classes de lignes** introduites par un partitionnement, il est nécessaire d'utiliser la commande **HAVING**.
Vous cherchez à sélectionner et visualiser pour chaque éditeur le nombre moyen de pages proposées. Seuls les éditeurs proposant une moyenne du nombre de pages supérieure à 83,1 devront être visualisés.

```
SELECT CodeEditeur, AVG(NbrePages)
FROM LIVRE
GROUP BY CodeEditeur
HAVING AVG(NbrePages) >83.1 ;
```

B) Les requêtes imbriquées

Elles sont appelées également sous-requêtes. Il s'agit d'une requête incorporée dans la commande WHERE ou HAVING d'une autre requête (requête principale). Cette dernière utilise les résultats de la sous-requête. Certaines sous-requêtes permettent de remplacer les jointures. Les sous-requêtes renvoient une ou plusieurs valeurs.

1- La requête ne renvoie qu'une seule valeur.

L'imbrication de la requête avec la sous-requête se fera avec un opérateur de comparaison (=, >, etc)

On souhaite connaître le code éditeur et le nom de l'éditeur du livre I101
La requête SQL peut alors être la suivante :

```
SELECT CodeEditeur, NomEditeur
FROM EDITEUR, LIVRE
WHERE CodeEditeur.EDITEUR=CodeEditeur.LIVRE
AND CodeLivre IN ('I101')
```

Elle peut également être la suivante :

```
SELECT CodeEditeur,NomEditeur
FROM EDITEUR
WHERE CodeEditeur =
                        (SELECT CodeEditeur
                         FROM  LIVRE
                        WHERE CodeLivre IN('I101')) ;
```

Remarque : La sous-requête est notée entre parenthèses.

Les requêtes imbriquées sont particulièrement adaptées avec des fonctions statistiques dans la sous-requête.
Exemple :
- Quel est le CodeLivre ayant le nombre de pages le plus élevé ?

```
SELECT CodeLivre
FROM LIVRE
WHERE  NbrePages =
                        (SELECT Max(NbrePages)
                         FROM LIVRE );
```

2- La sous-requête renvoie plusieurs valeurs

L'imbrication se fera avec bien souvent l'opérateur logique IN
Exemple :
On souhaite connaître le code editeur et le nom de l'éditeur des livres appelés INFO
La requête SQL peut alors être la suivante :

```
SELECT Distinct CodeEditeur, NomEditeur
FROM EDITEUR, LIVRE
WHERE CodeEditeur.EDITEUR=CodeEditeur.LIVRE
AND NomLivre IN('Info')
```

Elle peut également être la suivante :

```
SELECT CodeEditeur,NomEditeur
FROM EDITEUR
WHERE CodeEditeur IN
                        (SELECT CodeEditeur
                         FROM  LIVRE
                        WHERE NomLivre IN('Info')) ;
```

Dans la première formulation, la mention DISTINCT permet d'éliminer les doublons.
ORDER BY ne peut être noté dans une sous-requête.

II- La manipulation des données

Il est possible également de parler de mise à jour ou d'actualisation des données.

Pour cela, il existe principalement 3 commandes : INSERT, UPDATE et DELETE.

A) La commande INSERT

Elle permet d'insérer un ou plusieurs tuples dans une table. Il est possible de dire également que cette commande va permettre d'assurer le "peuplement" des tables de la base à partir de données nouvelles.
Le formalisme est le suivant :

```
INSERT INTO <TABLE>
VALUES ( ' valeur1 ',' valeur2 ') ;
```

Exemple :

```
INSERT INTO  EDITEUR
VALUES ('LACO1','BertrandLacopee', '8, rue de Nevers','75009','PARIS')
```

Si la mise à jour n'existe que pour quelques attributs, le formalisme est le suivant :

```
INSERT INTO <TABLE> (ATTRIBUT1, ATTRIBUT2)
VALUES ('valeur1','valeur2') ;
```

Il est alors nécessaire de respecter l'ordre des attributs.

Exemple :
On souhaite ajouter un livre. Deux éléments seulement sont connus le codelivre (F132) et le codeediteur (FOU1)

```
INSERT INTO LIVRE (CodeLivre, CodeEditeur)
VALUES ('F132','FOU1')
```

Dans tous les cas, les valeurs de la liste sont séparées par des virgules. Des guillemets simples doivent encadrer les données dès qu'elles comportent des caractères ou des dates. C'est inutile pour des données numériques ou pour des valeurs NULL (absence de valeurs).
Ainsi pour l'exemple précédent, il était également possible d'écrire :

```
INSERT INTO LIVRE
VALUES ('F132',",",'FOU1')
```

Ou bien encore, il était possible d'écrire :

```
INSERT INTO LIVRE
VALUES ('F132',NULL,NULL,FOU1)
```

Rque : L'absence de valeur pour un attribut (ou colonne) doit avoir été autorisée. C'est par exemple impossible pour la clé primaire.

B) La commande UPDATE

Elle permet de mettre à jour des données dans une table.
Le formalisme est le suivant :

```
UPDATE <table>
SET <attribut>=<' valeur '>
[WHERE <condition>] ;
```

Si le mot réservé WHERE est facultatif et permet de faire une mise à jour sous certaines conditions, il est très rare de ne pas le noter. L'inverse peut-être dangereux car l'ensemble de la colonne serait modifié.

Exemple : Le code éditeur a maintenant pour nom Foucher et Compagnie.

```
UPDATE EDITEUR
SET NomEditeur = 'Foucher et Compagnie'
WHERE CodeEditeur = 'FOU1'
```

S'il y a 2 mises à jour dans la même requête, il faut les séparer par une virgule et ne pas mettre 2 fois SET
La syntaxe est la suivante :

```
UPDATE <table>
SET     <attribut1>=<' valeur2 '>,
        <attribut2>=<' valeur4 '>
[WHERE <condition>] ;
```

Exemple : Le code éditeur a maintenant pour nom Fourcher et Compagnie et pour adresse 31 rue de Fleurus.

```
UPDATE EDITEUR
SET     NomEditeur = 'Fourcher et Compagnie'
        AdresseEditeur ='31, rue de Fleurus'
WHERE CodeEditeur = 'FOU1'
```

C) La commande DELETE

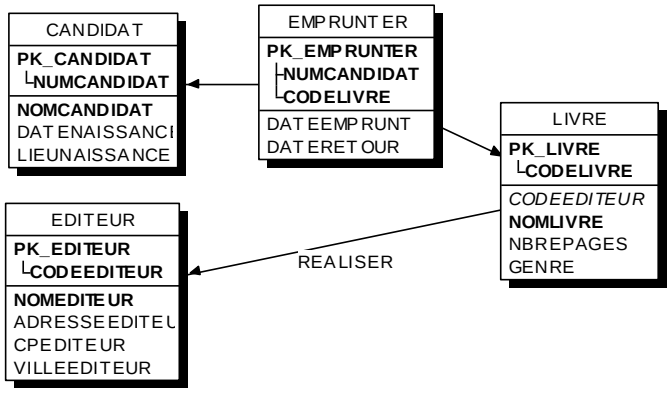
Elle permet d'effacer un ou plusieurs tuples.
DELETE FROM <table>
[WHERE <condition>] ;
Ici aussi le mot réservé WHERE est facultatif et permet de faire une suppression sous certaines conditions. Mais de la même manière, il est très rare de ne pas le noter. L'inverse peut-être dangereux car l'ensemble des enregistrements serait supprimé.

Exemple
On souhaite supprimer l'éditeur dont le code est F132.
La requête SQL est la suivante :

```
DELETE FROM EDITEUR
WHERE CodeEditeur IN('F132')
```

ANNEXE : BASE DE DONNEES BIBLIOTHEQUE

1- Sous forme graphique (sous WinDesign), le schéma relationnel est le suivant :



2- Les données

Liste des CANDIDATS

Numéro Candidat	Nom Candidat	Date Naissance	Lieu Naissance
12	BERTRAND	20/08/1978	RENNES
10	FAURE	12/12/1977	CAEN
8	HIRARD	20/12/1978	LILLE
15	FAURE	01/11/1977	ST LO

Liste des LIVRES

Code Livre	Nom Livre	Nbre de Pages	Genre	Code Editeur
I101	Info.	100	Informatique	FOU1
I102	Info.	123	Informatique	DUN1
F021	OMG	43	Gestion	DUN1
E120	FISCALITÉ	67	Gestion	FOU1

Liste des EDITEURS

Code Editeur	Nom Editeur	Adresse Editeur	CP Editeur	Ville Editeur
FOU1	Foucher	31, rue de Fleurus	75006	PARIS
DUN1	Dunod	14, rue de Prague	75008	PARIS
HA2	Hachette	5, rue d'Italie	75006	PARIS

Liste des EMPRUNTS

Numéro candidat	Code Livre	Date Emprunt	Date Retour
10	I101	1/10/2003	1/11/2003
8	I101	1/11/2003	12/11/2003
10	I102	14/12/2003	

REQUETES

A partir de la base de données BIBLIOTHEQUE, réaliser les requêtes suivantes en SQL :

Etape 1

01- Afficher la liste des livres en précisant toutes les caractéristiques.
02- Afficher la liste des livres et les afficher dans l'ordre décroissant des noms et en cas d'égalité dans l'ordre croissant des codes éditeurs.
03- Afficher pour chaque candidat les livres empruntés (Numéro Candidat, Code Livre).
04- Afficher pour chaque candidat les livres empruntés (Numéro Candidat, Code Livre, Nom Livre).
05- Afficher les codes des éditeurs dont au moins un livre a été emprunté. Chaque éditeur n'est noté qu'une seule fois.
06- Afficher les codes et noms des éditeurs dont au moins un livre a été emprunté. Chaque éditeur n'est noté qu'une seule fois.
07- Afficher les codes des livres qui ont été empruntés. Chaque livre n'est noté qu'une seule fois.
08- Pour chaque emprunt, afficher le code livre, le code éditeur, le numéro du candidat, le nom du candidat.
09- Afficher les caractéristiques de l'éditeur FOUCHER.
10- Afficher les livres (code livre, nom livre) qui ont 100 pages ou plus et qui ont au moins l'une des caractéristiques suivantes :
Caractéristique 1 : le code éditeur est FOU1
Caractéristique 2 : le nom du livre est INFO
11- Afficher toutes les caractéristiques des livres dont le nombre de pages est inférieur à 100.
12- Afficher les noms des livres et le nombre de pages des livres dont le nombre de pages est inférieur à 100. Les résultats seront affichés par ordre alphabétique.
13- Même question mais dans l'ordre alphabétique inverse.

Etape 2

14. Rechercher le numéro de candidat, le code livre et la date d'emprunt des livres non retournés.
15. Présenter les caractéristiques des livres dont le nombre de pages est supérieur à 50 pages mais inférieur à 100 pages.
16. Dans une seule requête, retrouver les caractéristiques des livres I101 et I102.
17. Retrouver toutes les caractéristiques des livres dont le nom commence par I.
18. Afficher toutes les caractéristiques des livres dont le deuxième caractère est la lettre M.
19. Afficher la liste des livres dont le nombre de pages est différent de 100.
20. Dans une seule requête, retrouver les caractéristiques de tous les livres sauf I101 et I102.
21. Afficher tous les livres dont la première lettre ne commence pas par I.
22. Présenter le(s) numéro(s) et le(s) nom(s) des candidats qui ont emprunté(s) un livre édité par l'éditeur FOUCHER et dont la première lettre du nom commence par H.

Etape 3

23. Déterminer le nombre de livres. La colonne sera appelée NbreLivres.
24. Afficher l'ensemble des caractéristiques des livres et ajouter une colonne supplémentaire qui donnera le nombre de pages +2 . Le nom de cette dernière colonne sera Nb2pages.
25. Calculer le nombre moyen de pages pour l'ensemble des livres.
26. Déterminer le nombre de livres empruntés pour chaque numéro de candidat.
27. Déterminer le nombre de livres empruntés pour chaque numéro et nom de candidat
28. Déterminer le nombre moyen de pages proposé par chaque code éditeur.
29. Déterminer le nombre moyen de pages proposé par chaque éditeur (code éditeur, nom éditeur, nombre moyen).
30. Indiquer le code, le nom des éditeurs dont le nombre moyen de pages est supérieur à 83. La moyenne des pages doit également être affichée.

Etape 4

31. Retrouver le livre (code livre, nom livre) ayant le nombre de pages le plus élevé.
32. Déterminer le livre (code livre, nom livre) ayant le nombre de pages le moins élevé.
33. Présenter la liste des livres (code livre, nom livre) ayant un nombre de pages supérieur à la moyenne du nombre de pages pour l'ensemble des livres.

PROPOSITION DE CORRIGE

2 remarques :
Ce sont des propositions. Ils existent parfois d'autres manières de faire.
Les requêtes ont été réalisées dans un éditeur SQL. Il peut y avoir quelques petites différences avec un travail sous Access (Exple : % au lieu de *).

Etape 1

01- Afficher la liste des livres en précisant toutes les caractéristiques.

```
-- Requête 01
SELECT *
FROM LIVRE ;
```

RESULTAT :

CODELIVRE;CODEEDITEUR;NOMLIVRE;NBREPAGES;GENRE
I101;FOU1;Info.;100;Informatique
I102;DUN1;Info.;123;Informatique
F021;DUN1;OMG;43;Gestion
E120;FOU1;FISCALITÉ;67;Gestion

02- Afficher la liste des livres et les afficher dans l'ordre décroissant des noms et en cas d'égalité dans l'ordre croissant des codes éditeurs.

```
-- Requête 02
SELECT CODELIVRE,CODEEDITEUR,NOMLIVRE,NBREPAGES,GENRE
FROM LIVRE
ORDER BY 3 DESC,2;
```

RESULTAT :

CODELIVRE;CODEEDITEUR;NOMLIVRE;NBREPAGES;GENRE
I102;DUN1;Info.;123;Informatique
I101;FOU1;Info.;100;Informatique
F021;DUN1;OMG;43;Gestion
E120;FOU1;FISCALITÉ;67;Gestion

03- Afficher pour chaque candidat les livres empruntés (Numéro Candidat, Code Livre).

```
-- Requête 03
SELECT NUMCANDIDAT,CODELIVRE
FROM EMPRUNTER;
```

RESULTAT :

NUMCANDIDAT;CODELIVRE
10;I101
8;I101
10;I102

04- Afficher pour chaque candidat les livres empruntés (Numéro Candidat, Code Livre, Nom Livre

```
-- Requête 04
SELECT NUMCANDIDAT,LIVRE.CODELIVRE, NOMLIVRE
FROM EMPRUNTER,LIVRE
WHERE EMPRUNTER.CodeLivre = LIVRE.CodeLivre ;
```

RESULTAT :

NUMCANDIDAT;CODELIVRE;NOMLIVRE
10;I101;Info.
8;I101;Info.
10;I102;Info.

05- Afficher les codes des éditeurs dont au moins un livre a été emprunté. Chaque éditeur n'est noté qu'une seule fois

```
-- Requête 05
SELECT DISTINCT CODEEDITEUR
FROM LIVRE, EMPRUNTER
WHERE EMPRUNTER.CodeLivre = LIVRE.CodeLivre;
```

RESULTAT :

CODEEDITEUR
DUN1
FOU1

06- Afficher les codes et noms des éditeurs dont au moins un livre a été emprunté. Chaque éditeur n'est noté qu'une seule fois.

```
 -- Requête 06
SELECT DISTINCT EDITEUR.CODEEDITEUR, NOMEDITEUR
FROM LIVRE, EMPRUNTER,EDITEUR
WHERE EMPRUNTER.CodeLivre = LIVRE.CodeLivre
AND LIVRE.CodeEditeur = EDITEUR.CodeEditeur ;
```

RESULTAT :

CODEEDITEUR;NOMEDITEUR
DUN1;Dunod
FOU1;Foucher

07- Afficher les codes des livres qui ont été empruntés. Chaque livre n'est noté qu'une seule fois.

```
-- Requête 07
SELECT DISTINCT LIVRE.CODELIVRE
FROM LIVRE, EMPRUNTER
WHERE EMPRUNTER.CodeLivre = LIVRE.CodeLivre ;
```

RESULTAT :

CODELIVRE
I101
I102

08- Pour chaque emprunt, afficher le code livre, le code éditeur, le numéro du candidat, le nom du candidat.

```
-- Requête 08
SELECT LIVRE.CODELIVRE, CodeEditeur, CANDIDAT.NumCandidat, NomCandidat
FROM LIVRE, EMPRUNTER, CANDIDAT
WHERE EMPRUNTER.CodeLivre = LIVRE.CodeLivre
AND EMPRUNTER.NumCandidat = CANDIDAT.NumCandidat ;
```

RESULTAT :

CODELIVRE;CodeEditeur;NumCandidat;NomCandidat
I101;FOU1;10;FAURE
I101;FOU1;8;HIRARD
I102;DUN1;10;FAURE

09- Afficher les caractéristiques de l'éditeur FOUCHER.

```
-- Requête 09
SELECT CODEEDITEUR,NOMEDITEUR,ADRESSEEDITEUR,CPEDITEUR,VILLEEDITEUR
FROM EDITEUR
WHERE NomEditeur IN('FOUCHER');
```

RESULTAT :

CODEEDITEUR;NOMEDITEUR;ADRESSEEDITEUR;CPEDITEUR;VILLEEDITEUR
FOU1;Foucher;31, rue de Fleurus;75006;PARIS

10- Afficher les livres (code livre, nom livre) qui ont 100 pages ou plus et qui ont au moins l'une des caractéristiques suivantes :
Caractéristique 1 : le code éditeur est FOU1
Caractéristique 2 : le nom du livre est INFO

```
-- Requête 10
SELECT CodeLivre,NomLivre
FROM LIVRE
WHERE NBREPAGES >= 100
AND (CODEEDITEUR = 'FOU1' OR NOMLIVRE = 'Info.');
```

RESULTAT :

CodeLivre;NomLivre
I101;Info.
I102;Info.

11- Afficher toutes les caractéristiques des livres dont le nombre de pages est inférieur à 100
```
-- Requête 11
SELECT CODELIVRE,CODEEDITEUR,NOMLIVRE,NBREPAGES,GENRE
FROM LIVRE
WHERE NBREPAGES <100;
```

RESULTAT :
CODELIVRE;CODEEDITEUR;NOMLIVRE;NBREPAGES;GENRE
F021;DUN1;OMG;43;Gestion
E120;FOU1;FISCALITÉ;67;Gestion

12- Afficher les noms des livres et le nombre de pages des livres dont le nombre de pages est inférieur à 100. Les résultats seront affichés par ordre alphabétique.
```
 -- Requête 12
SELECT NOMLIVRE,NBREPAGES
FROM LIVRE
WHERE NBREPAGES <100
ORDER BY 1;
```
RESULTAT :
NOMLIVRE;NBREPAGES
FISCALITÉ;67
OMG;43

13- Même question mais dans l'ordre alphabétique inverse
```
--Requête 13
SELECT NOMLIVRE,NBREPAGES
FROM LIVRE
WHERE NBREPAGES <100
ORDER BY 1 DESC ;
```
RESULTAT :
NOMLIVRE;NBREPAGES
OMG;43
FISCALITÉ;67

Etape 2

14- Rechercher le numéro de candidat, le code livre et la date d'emprunt des livres non retournés
```
--Requête 14
SELECT NUMCANDIDAT,CODELIVRE,DATEEMPRUNT
FROM EMPRUNTER
WHERE DATERETOUR IS NULL;
```
RESULTAT :
NUMCANDIDAT;CODELIVRE;DATEEMPRUNT
8;I101;1/11/2003

15- Présenter les caractéristiques des livres dont le nombre de pages est compris supérieur à 50 pages mais inférieur à 100 pages.
```
-- Requête 15
SELECT CODELIVRE,CODEEDITEUR,NOMLIVRE,NBREPAGES,GENRE
FROM LIVRE
WHERE NBREPAGES BETWEEN 51 AND 99;
```
RESULTAT :
CODELIVRE;CODEEDITEUR;NOMLIVRE;NBREPAGES;GENRE
E120;FOU1;FISCALITÉ;67;Gestion

16- Dans une seule requête, retrouver les caractéristiques des livres I101 et I102
```
-- Requête 16
SELECT CODELIVRE,CODEEDITEUR,NOMLIVRE,NBREPAGES,GENRE
FROM LIVRE
WHERE CodeLivre In ('I101','I102');
```
RESULTAT :
CODELIVRE;CODEEDITEUR;NOMLIVRE;NBREPAGES;GENRE
I101;FOU1;Info.;100;Informatique
I102;DUN1;Info.;123;Informatique

17- Retrouver toutes les caractéristiques des livres dont le nom commence par I
```
-- Requête 17
SELECT CODELIVRE,CODEEDITEUR,NOMLIVRE,NBREPAGES,GENRE
FROM LIVRE
WHERE NomLivre Like 'I%';
```
RESULTAT :
CODELIVRE;CODEEDITEUR;NOMLIVRE;NBREPAGES;GENRE
I101;FOU1;Info.;100;Informatique
I102;DUN1;Info.;123;Informatique

18- Afficher toutes les caractéristiques des livres dont le deuxième caractère est la lettre M.
```
-- Requête 18
SELECT CODELIVRE,CODEEDITEUR,NOMLIVRE,NBREPAGES,GENRE
FROM LIVRE
WHERE NomLivre Like '_M%' ;
```
RESULTAT :
CODELIVRE;CODEEDITEUR;NOMLIVRE;NBREPAGES;GENRE
F021;DUN1;OMG;43;Gestion

19- Afficher la liste des livres dont le nombre de pages est différent de 100.
```
-- Requête 19
SELECT CODELIVRE,CODEEDITEUR,NOMLIVRE,NBREPAGES,GENRE
FROM LIVRE
WHERE NBREPAGES <> 100;
```
RESULTAT :
CODELIVRE;CODEEDITEUR;NOMLIVRE;NBREPAGES;GENRE
I102;DUN1;Info.;123;Informatique
F021;DUN1;OMG;43;Gestion
E120;FOU1;FISCALITÉ;67;Gestion

20- Dans une seule requête, retrouver les caractéristiques de tous les livres sauf I101 et I102.
```
-- Requête 20
SELECT CODELIVRE,CODEEDITEUR,NOMLIVRE,NBREPAGES,GENRE
FROM LIVRE
WHERE CodeLivre NOT IN ('I101','I102');
```
RESULTAT :
CODELIVRE;CODEEDITEUR;NOMLIVRE;NBREPAGES;GENRE
F021;DUN1;OMG;43;Gestion
E120;FOU1;FISCALITÉ;67;Gestion

21Afficher tous les livres dont la première lettre ne commence pas par I.
```
-- Requête 21
SELECT CODELIVRE,CODEEDITEUR,NOMLIVRE,NBREPAGES,GENRE
FROM LIVRE
WHERE NomLivre NOT Like 'I%' ;
```
RESULTAT :
CODELIVRE;CODEEDITEUR;NOMLIVRE;NBREPAGES;GENRE
F021;DUN1;OMG;43;Gestion

RESULTAT :
E120;FOU1;FISCALITÉ;67;Gestion

22- Présenter le(s) numéro(s) et le(s) nom(s) des candidats qui ont emprunté(s) un livre édité par l'éditeur FOUCHER et dont la première lettre du nom commence par H.

```
-- Requête 22
SELECT CANDIDAT.NumCandidat, NomCandidat
FROM LIVRE, EMPRUNTER, CANDIDAT, EDITEUR
WHERE EMPRUNTER.CodeLivre = LIVRE.CodeLivre
AND EMPRUNTER.NumCandidat = CANDIDAT.NumCandidat
AND EDITEUR.CodeEditeur = LIVRE.CodeEditeur
AND NomEditeur IN('FOUCHER')
AND NomCandidat like 'H%';
```

RESULTAT :
NumCandidat;NomCandidat
8;HIRARD

Etape 3

23- Déterminer le nombre de livres. La colonne sera appelée NbreLivre.

```
-- Requête 23
SELECT COUNT(CodeLivre)AS NbreLivre
FROM LIVRE;
```

RESULTAT :
NbreLivre
4

24- Afficher l'ensemble des caractéristiques des livres et ajouter une colonne supplémentaire qui donnera le nombre de pages +2 . Le nom de cette dernière colonne sera Nb2pages.

```
-- Requête 24
SELECT CODELIVRE,CODEEDITEUR,NOMLIVRE,NBREPAGES,GENRE,NBREPAGES+2 AS Nb2pages
FROM LIVRE ;
```

RESULTAT :
CODELIVRE;CODEEDITEUR;NOMLIVRE;NBREPAGES;GENRE;Nb2pages
I101;FOU1;Info.;100;Informatique;102
I102;DUN1;Info.;123;Informatique;125
F021;DUN1;OMG;43;Gestion;45
E120;FOU1;FISCALITÉ;67;Gestion;69

25- Calculer le nombre moyen de pages pour l'ensemble des livres.

```
-- Requête 25
SELECT AVG(NBREPAGES)
FROM LIVRE ;
```

RESULTAT :
Expr1000
83,25

26- Déterminer le nombre de livres empruntés pour chaque numéro de candidat.

```
-- Requête 26
SELECT NumCandidat,COUNT(CodeLivre)AS NbreLivre
FROM EMPRUNTER
GROUP BY NumCandidat ;
```

RESULTAT :
NumCandidat;NbreLivre
10;2
8;1

27-Déterminer le nombre de livres empruntés pour chaque numéro et nom de candidat

```
-- Requête 27
SELECT CANDIDAT.NumCandidat,NomCandidat,COUNT(CodeLivre)AS NbreLivre
FROM EMPRUNTER, CANDIDAT
WHERE EMPRUNTER.NumCandidat = CANDIDAT.NumCandidat
GROUP BY CANDIDAT.NumCandidat, NomCandidat ;
```

RESULTAT :

NumCandidat;NomCandidat;NbreLivre
10;FAURE;2
8;HIRARD;1

28- Déterminer le nombre moyen de pages proposé par chaque code éditeur.

```
-- Requête 28
SELECT CodeEditeur, AVG(NbrePages)
FROM LIVRE
GROUP BY CodeEditeur ;
```

RESULTAT :

CodeEditeur;Expr1001
DUN1;83
FOU1;83,5

29- Déterminer le nombre moyen de pages proposé par chaque éditeur (code éditeur, nom éditeur, nombre moyen).

```
-- Requête 29
SELECT EDITEUR.CodeEditeur,NomEditeur,AVG(NbrePages)AS MOYNBPAGES
FROM LIVRE,EDITEUR
WHERE LIVRE.CodeEditeur = EDITEUR.CodeEditeur
GROUP BY EDITEUR.CodeEditeur,NomEditeur ;
```

RESULTAT :

CodeEditeur;NomEditeur;MOYNBPAGES
DUN1;Dunod;83
FOU1;Foucher;83,5

30- Indiquer le code, le nom des éditeurs dont le nombre moyen de pages est supérieur à 83. La moyenne des pages doit également être affichée.

```
-- Requête 30
SELECT EDITEUR.CodeEditeur,NomEditeur,AVG(NbrePages)AS MOYNBPAGES
FROM LIVRE,EDITEUR
WHERE LIVRE.CodeEditeur = EDITEUR.CodeEditeur
GROUP BY EDITEUR.CodeEditeur,NomEditeur
HAVING AVG(NbrePages)>83  ;
```

RESULTAT :

CodeEditeur;NomEditeur;MOYNBPAGES
FOU1;Foucher;83,5

Etape 4

31- Retrouver le livre (code livre, nom livre) ayant le nombre de pages le plus élevé.

```
-- Requête 31
SELECT CodeLivre, NomLivre
FROM LIVRE
WHERE NBREPAGES =(SELECT Max(NbrePages) FROM LIVRE );
```

RESULTAT :

CodeLivre;NomLivre
I102;Info.

32- Déterminer le livre (code livre, nom livre) ayant le nombre de pages le moins élevé.

```
-- Requête 32
SELECT CodeLivre, NomLivre
FROM LIVRE
WHERE NBREPAGES =(SELECT Min(NbrePages) FROM LIVRE ) ;
```

RESULTAT :

CodeLivre;NomLivre
F021;OMG

33- Présenter la liste des livres (code livre, nom livre) ayant un nombre de pages supérieur à la moyenne du nombre de pages pour l'ensemble des livres.

```
SELECT CodeLivre, NomLivre
FROM LIVRE
WHERE NBREPAGES >(SELECT AVG(NbrePages) FROM LIVRE );
```

RESULTAT :

CodeLivre;NomLivre
I101;Info.
I102;Info.

LEXIQUE présenté dans le cadre du sujet de métropole en 2003 pour le BTS CGO

PROJECTION D'ATTRIBUTS		
Expression	**Résultat**	**Syntaxe**
SELECT	Spécifie les attributs que l'on veut extraire et afficher	SELECT TABLE.Attribut
FROM	Spécifie les tables nécessaires à la requête.	FROM TABLE1, TABLE2
;	Indique que la requête est terminée	**;**
SELECTION DE TUPLES		
Expression	**Résultat**	**Syntaxe**
WHERE	Précède la première jointure ou sélection	**WHERE TABLE.Attribut** LIKE chaîne de caractères
AND	Succède à WHERE que ce soit pour une sélection ou une jointure	**AND TABLE.Attribut** = Valeur numérique
OR	Précède une sélection (union)	**OR TABLE.Attribut** = Valeur numérique
LIKE / =	LIKE précède une chaîne de caractères. = précède une valeur numérique.	WHERE TABLE.Attribut **LIKE chaîne de caractères** AND TABLE.Attribut **= Valeur numérique**
IS [NOT] NULL	Prédicat de [non] nullité.	**WHERE** TABLE.Attribut IS [NOT] NULL
BETWEEN … AND …	Prédicat d'intervalle. Equivalent à >= … AND <= …	WHERE **TABLE.Attribut BETWEEN valeur1 AND valeur 2**
TRI		
Expression	**Résultat**	**Syntaxe**
ORDER BY … ASC ou DESC	La hiérarchie des clés de tri est définie par l'ordre des attributs derrière ORDER BY	ORDER BY TABLE.Attribut1, TABLE.Attribut2 ASC
INTERSECTION		
Expression	**Résultat**	**Syntaxe**
IN	Permet une intersection.	**IN (Requête)**
NOT IN	Permet de faire une différence.	**NOT IN (Requête)**
CALCULS		
Expression	**Résultat**	**Syntaxe**
SUM	Retourne la somme des valeurs d'un attribut d'une table	**SELECT SUM (TABLE.Attribut)** AS NomAlias
AVG	Retourne la moyenne des valeurs d'un attribut d'une table	**SELECT AVG (TABLE.Attribut)** AS NomAlias
MAX	Retourne la valeur maximum d'un attribut d'une table	**SELECT MAX (TABLE.Attribut)** AS NomAlias
MIN	Retourne la valeur minimum d'un attribut d'une table	**SELECT MIN (TABLE.Attribut)** AS NomAlias
AS	L'attribut projeté est identifié par le nom de l'alias	SELECT SUM (TABLE.Attribut) **AS NomAlias**
REGROUPEMENT		
Expression	**Résultat**	**Syntaxe**
COUNT	Retourne le nombre de tuples d'une table	**SELECT COUNT (TABLE.Attribut)** AS NomAlias
GROUP BY	Permet de faire porter les fonctions d'agrégat sur des partitions de la table.	GROUP BY TABLE.Attribut HAVING TABLE.Attribut = Valeur
HAVING	Permet d'appliquer des prédicats de condition sur des résultats de regroupement.	GROUP BY TABLE.Attribut **HAVING TABLE.Attribut** = Valeur
MISE A JOUR		
Expression	**Résultat**	**Syntaxe**
INSERT INTO	Permet d'insérer ou ou plusieurs tuples dans une table.	**INSERT INTO TABLE.Attribut1, TABLE.Attribut2**
VALUES	Précise les valeurs que l'on va attribuer aux tuples à insérer	**VALUES (valeur1, valeur2)**
UPDATE	Modifie la valeur d'un ou plusieurs attributs dans un ou plusieurs tuples d'une table	UPDATE TABLE
SET	Donne les nouvelles valeurs	SET TABLE.Attribut = Nouvelle valeur
DELETE FROM	Efface un ou plusieurs tuples de la table	DELETE FROM TABLE