Bases de données et SQL

1 Formalisme et vocabulaire d'une base de données

1.1 Une table

Une base de données se présente comme un ensemble de tableaux. Dans ce contexte un tableau sera nommé **table** (on parle aussi de *relation*, si on veut faire de l'algèbre relationnelle qui est le background théorique des bases de données. Mais ici ce n'est pas l'objet).

Si on considère une base de données d'une librairie, on aura par exemple un tableau recensant les livres, qu'on nommera habilement `Livres`.
Considérons par exemple une table dont les colonnes sont les suivantes :

Titre	Auteur	Nombre de pages	Prix
...	...	...	...

Dans cette table, on appelle **enregistrement** une ligne du tableau (cela correspond donc ici à un livre).
Ce livre possède diverses caractéristiques (titre, auteur,...) nommées **attributs**. Chaque attribut a un certain format (texte, nombre entier, date...). Ceci définit le **domaine** de l'attribut. Ces domaines sont à déclarer à la création de la table. Les deux plus courants sont le format texte (`TEXT`, ou `VARCHAR(n)` pour limiter la longueur du texte à n caractères) et le format nombre entier naturel (`INTEGER`).
Un attribut correspond donc à une colonne du tableau. On parle aussi de **champ**. On a aussi le format nombre à virgule : `FLOAT`.
Ajoutons donc quelques livres :

Titre	Auteur	Nombre de pages	Prix
Ubik	Dick	288	15.9
Salammbô	Flaubert	544	14.9
Madame Bovary	Flaubert	672	18.5
Le procès	Kafka	280	25

NB : dans le format de base de données qu'on décrit ici, il n'y a pas de notion d'ordre entre les lignes. Par contre on verra plus tard qu'on peut décider d'ordonner les résultats d'une requête suivant certaines caractéristiques.

1.2 Clé primaire

Un des enjeux dans la constitution d'une base de données est de pouvoir identifier un livre rapidement sans avoir à énumérer toutes ses caractéristiques. Ici on voit que le couple (Titre,Auteur) pourrait permettre une identification non ambiguë... mais certains livres existent en plusieurs éditions !

On appelle *clé primaire* un attribut, ou un ensemble d'attributs, qui identifient de manière unique chaque enregistrement. Avec la remarque précédente, si on limite notre table à ces 4 livres, le couple (Titre,Auteur) est une clé primaire ; mais l'évolution ultérieure de notre table ne nous garantit pas que ce sera toujours le cas. La démarche habituelle est donc de **créer un attribut « artificiel » dont le seul but sera d'être une clé primaire** (dans ce qui suit, l'attribut `id`). Dit en français : on donne un numéro à chaque enregistrement.

On modifie donc la table précédente en :

id	Titre	Auteur	Nombre de pages	Prix
1	Ubik	Dick	288	15.9
2	Salammbô	Flaubert	544	14.9
3	Madame Bovary	Flaubert	672	18.5
4	Le procès	Kafka	280	25

et l'attribut `id` (nom traditionnel, mais on peut le nommer comme on veut !) est cette fois une clé primaire. Il devra être déclaré comme tel lors de la création de la table en SQL (cf plus bas).

Ainsi, à d'autres endroits de la base de données, on pourra faire référence au « Livre 1 » , sans avoir à préciser qu'il s'agit d'Ubik, de Philip K. Dick, ayant 288 pages et coûtant 15.9€.
Ceci doit en fait être déclaré dans le schéma de table : ainsi le gestionnaire de base de données s'assurera, au cours de l'évolution des données enregistrées, que les valeurs prises par la clé primaire sont toujours 2 à 2 distinctes.

1.3 Lien entre plusieurs tables : clés étrangères

Une base de données est un ensemble de plusieurs tables. Si on est une librairie, on peut avoir une table `Livres`, une table `Clients`, une table `Achats`, une table `Stocks`, etc...
Plus intéressant : on peut mettre les tables en relation. Si Ubik est identifié par la valeur « 1 » de la clé primaire `id` dans la table `Livres`, on peut, dans la table répertoriant les achats, signaler que « le Livre 1 » a été acheté, sans avoir à répéter toutes les informations qui lui sont associées. Supposons donc qu'on enregistre les achats effectués dans la librairie avec une table `Achats`, qui recense le nom du client et le livre acheté :

Nom	Prénom	Livre	Date
Hollande	François	2	2018-02-12
Sarkozy	Nicolas	4	2018-03-27
Trump	Donald	4	2018-05-01

On voit ici que le nombre à renseigner dans la colonne `Livre` doit correspondre à un livre existant ; et donc être une valeur qui apparaît dans la colonne `id` de la table `Livres`. Pour qu'un tel procédé fonctionne de manière non ambiguë on voit que l'attribut auquel font référence les nombres de la colonne Livre doit être une clé primaire : s'il y a deux « Livre 1 » , on aura des soucis !

La colonne `Livre` de la table `Achats` est alors appelée **clé étrangère**. D'après la remarque précédente,

les valeurs contenues dans la clé étrangère feront toujours référence à une clé primaire.

À la création d'une colonne comportant une clé étrangère, on doit signaler cette propriété ainsi que la clé primaire à laquelle elle fait référence. Le gestionnaire de bases de données s'assurera alors, à chaque ajout d'un enregistrement dans la table `Achats`, que la valeur renseignée dans la colonne `Livre` est effectivement une valeur prise par la clé primaire `id` de la table `Livres`.
Avec les tables ci-dessus, l'ajout de l'enregistrement

Macron	Emmanuel	6	2018-07-07

à la table `Achats` renverra une erreur car il n'existe pas de livre ayant un `id` égal à 6 dans la table `Livres`.

On déclare une clé étrangère à la création de la table en spécifiant à quelle clé primaire elle fait référence. Le schéma de la table `Achats` sera donc ici :

```
Achats = (Nom:TEXT),(Prénom:TEXT),(FOREIGN KEY Livre REFERENCES Livres.id),(Date:DATE)
```

2 Construire et modifier une base de données avec SQL

Le langage SQL permet de piloter une base de données (créer les tables, les remplir, les modifier, les effacer, et en extraire de l'information). Voyons ici comment se font les opérations permettant de créer les tables définies dans la section précédente.

La tradition veut que les mots-clés du langage soient écrits en MAJUSCULES. Toutefois le programme n'est pas sensible à la casse (*case-sensitive* en anglais), y compris les noms des tables et des attributs.
Les retours à la ligne ne sont pas interprétés par le langage : on peut donc tout écrire sur une ligne, ou aller à la ligne autant de fois qu'on le souhaite, indifféremment.

2.1 Créer une table

Le mot clé est `CREATE TABLE`.
Il faut spécifier :

- Le nom de la table
- Une liste avec des titres de colonne (les attributs), le format de données, et des propriétés (clé primaire, clé étrangère avec la clé primaire à laquelle on fait référence)

Dans le contexte évoqué précédemment, on aura donc :

```
1  CREATE TABLE Livres (
2  id  INTEGER PRIMARY KEY,
3  titre  TEXT,
4  auteur  TEXT,
5  nombre_de_pages  INTEGER,
6  prix  FLOAT
7  )
```

Attention à la position des virgules : elles séparent les différents attributs, mais il n'y en a pas à la fin (donc ici pas de virgule après `prix FLOAT`).
Remarquer ces erreurs[1] car SQL est *extrêmement mauvais* pour expliquer les erreurs de syntaxe !

```
1  CREATE TABLE Achats (
2  nom  TEXT,
3  prenom  TEXT,
4  Livre REFERENCES Livres(id),
5  date  DATE
6  )
```

Noter ici à la ligne 4 la référence à un attribut d'une autre table par la syntaxe `nom_table(attribut)`. De plus il est inutile de préciser le domaine d'une clé étrangère : c'est obligatoirement celui de la clé primaire à laquelle elle fait référence !
Attention ceci est la syntaxe qui fonctionne dans SQLiteStudio ; mais dans le programme officiel d'ECG (et donc sur les copies) on conseille plutôt la syntaxe suivante :

```
1  ...
2  ...
3  ...
4  FOREIGN KEY Livre REFERENCES Livres(id),
5  ...
6  ...
```

[1] On peut néanmoins espérer qu'elles ne seront pas trop pénalisées aux concours !

2.2 Ajouter des lignes à une table

On insère dans une table (`INSERT INTO`) des valeurs (`VALUES`) correspondant à tout ou partie des attributs.
Ne pas oublier qu'une donnée de type `TEXT` (chaîne de caractères) est délimitée par des apostrophes `' '`.

```
INSERT INTO Livres
VALUES (1,'Ubik' , 'Dick' , 288 , 15.9)
```

ou si on veut seulement un sous-ensemble d'attributs :

```
INSERT INTO Livres (titre,auteur)
VALUES ('Salammbô','Flaubert')
```

Les attributs non renseignés sont alors vides (`NULL`).
On peut ajouter plusieurs lignes séparées par une virgule :

```
INSERT INTO Livres
VALUES
(2,'Salammbô' , 'Flaubert' , 544 , 14.9),
(3,'Madame Bovary' , 'Flaubert' , 672 , 18.5),
(4,'Le procès' , 'Kafka' , 280 , 25)
```

NB : si une des colonnes est une clé primaire, on peut, à la création de la table, passer une option `AUTOINCREMENT`.
La commande de création de table est modifiée en :

```
CREATE TABLE Livres (
id  INTEGER PRIMARY KEY AUTOINCREMENT,
.....
```

À l'ajout d'un enregistrement on n'est alors pas obligé de donner une valeur pour la clé primaire : le programme assigne automatiquement la valeur suivante.
Si on essaie de donner à l'attribut clé primaire une valeur déjà attribuée, le système renverra une erreur (`UNIQUE constraint failed`).

2.3 Enlever des lignes à une table

Il faut spécifier un critère définissant les lignes à enlever. On introduit ici le mot-clé `WHERE`, qui permettra de désigner des enregistrements vérifiant une ou plusieurs conditions.
Si on veut effacer dans la table les enregistrements correspondant aux livres de Flaubert, on tapera :

```
DELETE FROM Livres
WHERE auteur='Flaubert'
```

Si on ne précise pas de condition, on vide la table (mais sans l'effacer)

```
DELETE FROM Livres
```

Et si on veut supprimer la table :

```
DROP TABLE Livres
```

2.4 Enlever / ajouter des colonnes à une table

C'est un peu plus compliqué car on touche au schéma de la table en modifiant la liste des attributs. Pour ajouter une colonne de pays d'origine de l'auteur (en format « texte ») à notre table de livres :

```
ALTER TABLE Livres
ADD pays_auteur TEXT
```

puis l'enlever

```sql
ALTER TABLE Livres
DROP pays_auteur
```

2.5 Modifier le contenu de cellules existantes

Dans l'exemple ci-dessous, on modifie à l'aide du mot-clé `WHERE` les attributs du livre d'identifiant 1.

```sql
UPDATE Livres
SET titre = 'The Man in the High Castle',
nombre_de_pages = 348
WHERE id = 1
```

3 Lire des données dans une table avec SQL

3.1 Affichage (`SELECT`) et filtrage (`WHERE`)

La commande `SELECT` sélectionne (et affiche) les données correspondant à un jeu de conditions.
Comme assez souvent en informatique, l'étoile * désigne « tout » . On peut alors afficher toute la table :

```sql
SELECT * FROM Livres
```

Mais de manière générale on demande d'afficher un sous-ensemble d'attributs des enregistrements vérifiant certaines conditions. Ces conditions se situent derrière le mot-clé `WHERE` ; on peut les enchaîner par les opérateurs logiques usuels `AND`, `OR` et `NOT`.
Avec la table de livres créée plus haut, on peut :

Afficher les lignes complètes correspondant aux livres de Flaubert :

```sql
SELECT * FROM Livres WHERE auteur='Flaubert'
```

On obtient

id	Titre	Auteur	Nombre de pages	Prix
2	Salammbô	Flaubert	544	14.9
3	Madame Bovary	Flaubert	672	18.5

Afficher les nombres de pages des livres de Dick (ici on ne demande qu'un seul attribut, d'où le `SELECT nombre_de_pages`)

```sql
SELECT nombre_de_pages FROM Livres WHERE auteur='Dick'
```

On obtient :

Nombre de pages
288

On peut aussi imposer des contraintes numériques : ici on affiche les titres, prix, et nombres de pages des livres de plus de 400 pages :

```sql
SELECT titre,prix,nombre_de_pages FROM Livres
WHERE nombre_de_pages >=400
```

et on obtient

Titre	Prix	Nombre de pages
Salammbô	14.9	544
Madame Bovary	18.5	672

(NB : on peut afficher les attributs dans un ordre quelconque).
On dispose des tests numériques habituels : =, >, <, >=, <=, <> (ce dernier signifiant : « différent de »).

On peut enchaîner les tests logiques : ici on demande les livres de Flaubert dont le prix est supérieur à 16 euros.

```
1  SELECT * FROM Livres WHERE auteur='Flaubert' AND prix > 16
```

On trouve bien :

id	Titre	Auteur	Nombre de pages	Prix
3	Madame Bovary	Flaubert	672	18.5

La commande `DISTINCT` renvoie les valeurs distinctes prises par un attribut. Ainsi

```
1  SELECT DISTINCT auteur FROM Livres
```

renverra la liste des auteurs des livres de la table, sans répétition. On obtient

Auteur
Dick
Flaubert
Kafka

3.2 Outils numériques et fonctions d'agrégation

Le langage SQL permet de faire des statistiques rudimentaires sur les colonnes chiffrées : ce sont les *fonctions d'agrégation* `MIN`, `MAX`, `SUM`, `AVG` (moyenne), `COUNT` (nombre d'éléments).

Calculons par exemple la moyenne des prix de tous les livres :

```
1  SELECT AVG(prix) FROM Livres
```

et la somme des prix des livres de Flaubert :

```
1  SELECT SUM(prix) FROM Livres WHERE auteur='Flaubert'
```

On peut effectuer des opérations arithmétiques sur les valeurs numériques d'une table. Par exemple, un lecteur soucieux d'en avoir pour son argent pourrait se demander quel est le livre pour lequel le prix de la page est le moins cher. Il peut alors interroger :

```
1  SELECT titre,prix/nombre_de_pages FROM Livres
```

(le caractère / désigne donc ici une division) pour obtenir

titre	prix/nombre de pages
Ubik	0.05525633333333
Salammbô	0.02738970588235
Madame Bovary	0.02752976190476
Le procès	0.08928571428571

On peut également classer les résultats avec la commande `ORDER BY`. Pour avoir la liste de tous les livres, classés par prix croissant[2]

```
SELECT * FROM Livres  ORDER BY prix
```

renvoie

id	Titre	Auteur	Nombre de pages	Prix
2	Salammbô	Flaubert	544	14.9
1	Ubik	Dick	288	15.9
3	Madame Bovary	Flaubert	672	18.5
4	Le procès	Kafka	280	25

Si on veut classer par ordre décroissant, il faut ajouter l'option `DESC` :

```
SELECT * FROM Livres ORDER BY prix DESC
```

On peut également compter les occurrences d'une propriété dans une table, avec la commande `COUNT`. On peut par exemple se demander combien de livres valent (strictement) plus de 17€. Pour ce faire on écrit :

```
SELECT COUNT(*) FROM Livres WHERE prix>17
```

Exercice : comment notre lecteur près de ses sous peut-il renvoyer la liste des livres classés par prix à la page croissant ?

4 Jointure

Une des principales caractéristiques des bases de données relationnelles est de pouvoir *mettre en relation* les données contenues dans plusieurs tables.
Reprenons notre table `Livres` :

id	Titre	Auteur	Nombre de pages	Prix
1	Ubik	Dick	288	15.9
2	Salammbô	Flaubert	544	14.9
3	Madame Bovary	Flaubert	672	18.5
4	Le procès	Kafka	280	25

et complétons la table recensant les achats effectués dans la librairie (nommée `Achats`) :

Nom	Prénom	Livre	Date
Hollande	François	2	2018-02-12
Sarkozy	Nicolas	4	2018-03-27
Trump	Donald	4	2018-05-01
Macron	Emmanuel	1	2018-07-07
Hollande	François	1	2018-12-12

En croisant les informations des deux tables, il est clair que François Hollande a acheté Salammbô et Ubik. Pour cela, on consulte la table des achats, on récupère les numéros des livres achetés (2 et 1), et on regarde à quoi correspondent ces numéros dans la table des livres.
Il faut donc identifier l'attribut « `Livre` » de la table `Achats` et l'attribut « `id` » de la table `Livres`. C'est le principe d'une jointure.

À la base, une jointure se comporte comme un produit cartésien. La jointure des tables `Livres` et `Achats` produit une table à $5+4=9$ colonnes, dont les lignes sont toutes les réunions possibles d'une ligne de `Livres` et une ligne de `Achats`. C'est assez gros (c'est le premier tableau de la page finale), et on se rend compte que ça ne sert à rien. Les lignes intéressantes de ce tableau sont celles constituées d'un achat, et des informations sur le livre *auquel fait référence cet achat.* Autrement dit, on veut conserver les lignes pour lesquelles la valeur dans la colonne « `Livre` » de la table `Achats` est égale à celle dans la colonne « `id` » de la table `Livres`.

[2] C'est l'option de classement par défaut.

Il reste alors les lignes indiquées dans le second tableau de la page finale ; et ici le livre acheté par chaque client apparaît clairement.
La jointure s'effectue par la commande `INNER JOIN` ; les conditions à imposer (celles qui permettent de ne garder que les lignes intéressantes) se déclarent derrière un mot-clé `ON`.
Le gros tableau est obtenu par

```
1 SELECT * FROM Livres INNER JOIN Achats
```

alors que le petit, dans lequel on a égalé les deux colonnes identifiant les livres, sera donné par

```
1 SELECT * FROM Livres INNER JOIN Achats ON Livres.id = Achats.livre
```

Comme on gère maintenant plusieurs tables à la fois, il est nécessaire d'avoir une syntaxe qui donne un attribut en faisant référence à la table à laquelle il appartient. La syntaxe est `table.attribut`.
Noter que si le nom de l'attribut n'est pas ambigu (ici, un seul attribut s'appelle `id`), il n'est pas indispensable de spécifier la table à laquelle il appartient : la syntaxe

```
1 SELECT * FROM Livres JOIN Achats ON id = Achats.livre
```

fonctionne également.
Ensuite il ne reste plus qu'à extraire l'information voulue de cette jointure. Par exemple, pour obtenir les titres et prix des livres achetés par François Hollande, on tapera :

```
1 SELECT Livres.titre,Livres.prix
2 FROM Livres JOIN Achats ON Livres.id = Achats.livre
3 WHERE Achats.nom='Hollande'
```

On peut ensuite effectuer une jointure de trois tables ou plus... (cf. exercices)

id	Titre	Auteur	Nombre de pages	Prix	Nom	Prénom	Livre	Date
1	Ubik	Dick	288	15.9	Hollande	François	2	2018-02-12
1	Ubik	Dick	288	15.9	Sarkozy	Nicolas	4	2018-03-27
1	Ubik	Dick	288	15.9	Trump	Donald	4	2018-05-01
1	Ubik	Dick	288	15.9	Macron	Emmanuel	1	2018-07-07
1	Ubik	Dick	288	15.9	Hollande	François	1	2018-12-12
2	Salammbô	Flaubert	544	14.9	Hollande	François	2	2018-02-12
2	Salammbô	Flaubert	544	14.9	Sarkozy	Nicolas	4	2018-03-27
2	Salammbô	Flaubert	544	14.9	Trump	Donald	4	2018-05-01
2	Salammbô	Flaubert	544	14.9	Macron	Emmanuel	1	2018-07-07
2	Salammbô	Flaubert	544	14.9	Hollande	François	1	2018-12-12
3	Madame Bovary	Flaubert	672	18.5	Hollande	François	2	2018-02-12
3	Madame Bovary	Flaubert	672	18.5	Sarkozy	Nicolas	4	2018-03-27
3	Madame Bovary	Flaubert	672	18.5	Trump	Donald	4	2018-05-01
3	Madame Bovary	Flaubert	672	18.5	Macron	Emmanuel	1	2018-07-07
3	Madame Bovary	Flaubert	672	18.5	Hollande	François	1	2018-12-12
4	Le procès	Kafka	280	25	Hollande	François	2	2018-02-12
4	Le procès	Kafka	280	25	Sarkozy	Nicolas	4	2018-03-27
4	Le procès	Kafka	280	25	Trump	Donald	4	2018-05-01
4	Le procès	Kafka	280	25	Macron	Emmanuel	1	2018-07-07
4	Le procès	Kafka	280	25	Hollande	François	1	2018-12-12

id	Titre	Auteur	Nombre de pages	Prix	Nom	Prénom	Livre	Date
1	Ubik	Dick	288	15.9	Macron	Emmanuel	1	2018-07-07
1	Ubik	Dick	288	15.9	Hollande	François	1	2018-12-12
2	Salammbô	Flaubert	544	14.9	Hollande	François	2	2018-02-12
4	Le procès	Kafka	280	25	Sarkozy	Nicolas	4	2018-03-27
4	Le procès	Kafka	280	25	Trump	Donald	4	2018-05-01