Langage SQL
Création et modification d'une base de données

1 Cahier des charges

Avant de créer une base de données (de même pour tout projet informatique) il faut faire ce que l'on appelle le cahier des charges, c'est à dire faire la liste de tous les problèmes à résoudre

Dans le cadre de la création d'une base de données il faut partir des requêtes que l'on aimerait pouvoir faire sur cette base de données

Prenons un exemple : la création d'une base de données pour une bibliothèque municipale :

En tant qu'**emprunteur** j'aimerais pouvoir interroger la base pour savoir si :

1. le livre "Petits poèmes en prose" de Beaudelaire est dans la bibliothèque
2. En combien d'exemplaires ? En combien d'exemplaires disponibles ?Dans quel rayon ?
3. Quels livres de Dostoievski sont disponibles ?
4. Dans combien de temps dois je rendre les livres que j'ai empruntés ?

En tant qu'**administrateur** j'aimerais pouvoir interroger la base pour savoir si :

1. Quels sont les livres "en retard" ?
2. Quels sont les livres "en retard" de plus de trois mois ?
3. Quels sont les emprunteurs qui doivent renouveler leur inscription ?

En tant qu'**administrateur** j'aimerais pouvoir **mettre à jour** la base pour :

1. insérer de nouveaux livres et de nouveaux exemplaires
2. mettre à jour l'abonnement d'un emprunteur
3. changer l'adresse d'un emprunteur
4. Supprimer les prêts d'un emprunteur concernant un certain nombre d'exemplaires rendus

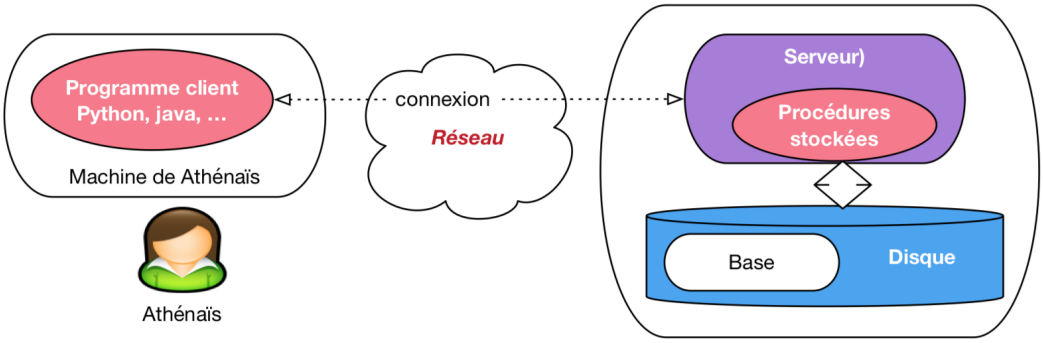
Au fur et à mesure que le cahier des charges s'affine on se rend compte des relations ou tables nécessaires avec leurs attributs, leurs clés primaires et étrangères (dépendance fonctionnelle) pour cette bibliothèque municipale

Une bibliothèque peut être vue comme des **Emprunteurs** réalisant des **Prêts d'Exemplaires** de **Livres** écrits par des **Auteurs** et édités par des **Editeurs**

Voici un schéma relationnel possible :

1. Emprunteur(**idAbonné**,nom,prénom,adresse,email,dateAbonnement)
2. Auteur(**idAuteur**,nom,prénom)
3. Editeur(**idEditeur**,nom)
4. Livre(**codeLivre**,titre,*idAuteur*)
5. Exemplaire(**codeRayon**,titre,*idLivre*,*idEditeur*)
6. Prêt(***codeAbonné,codeExemplaire***,datePrêt,dateRetour)

2 Création

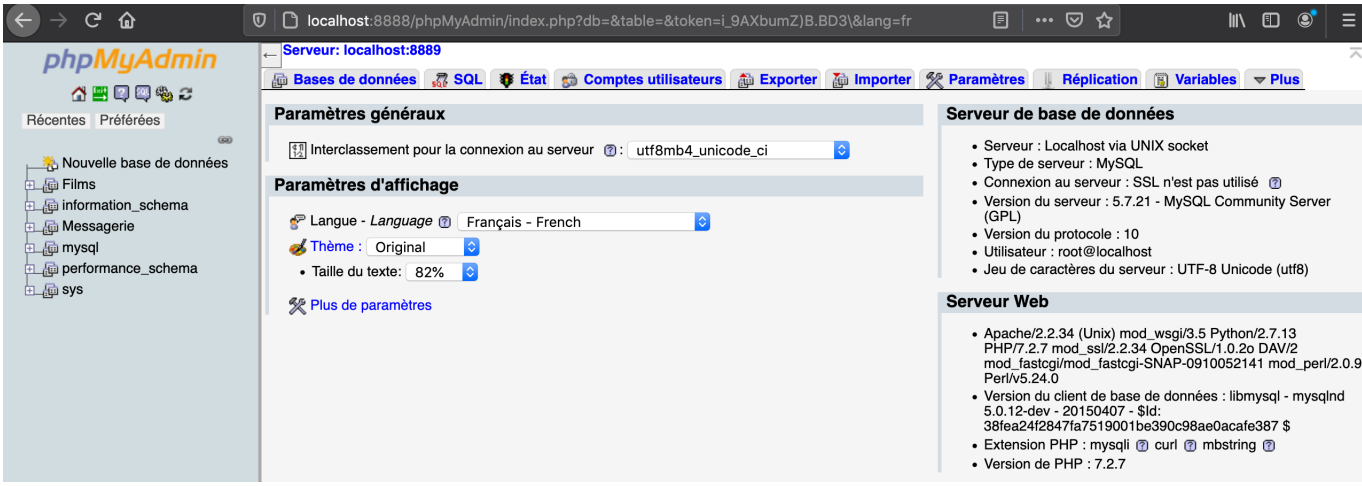


(Source : Cours de Philippe Rigaux sur les bases de données au CNAM)

Il existe deux façons de créer une base de données, soit **par programmation** par exemple avec Python et SQL ou SQLite, soit avec un **outil graphique** comme phpMyAdmin (SQL) ou SQLITE Browser

Nous allons créer la base de données une première fois avec phpMyAdmin puis une seconde fois avec Python et SQL

3 *phpMyAdmin*



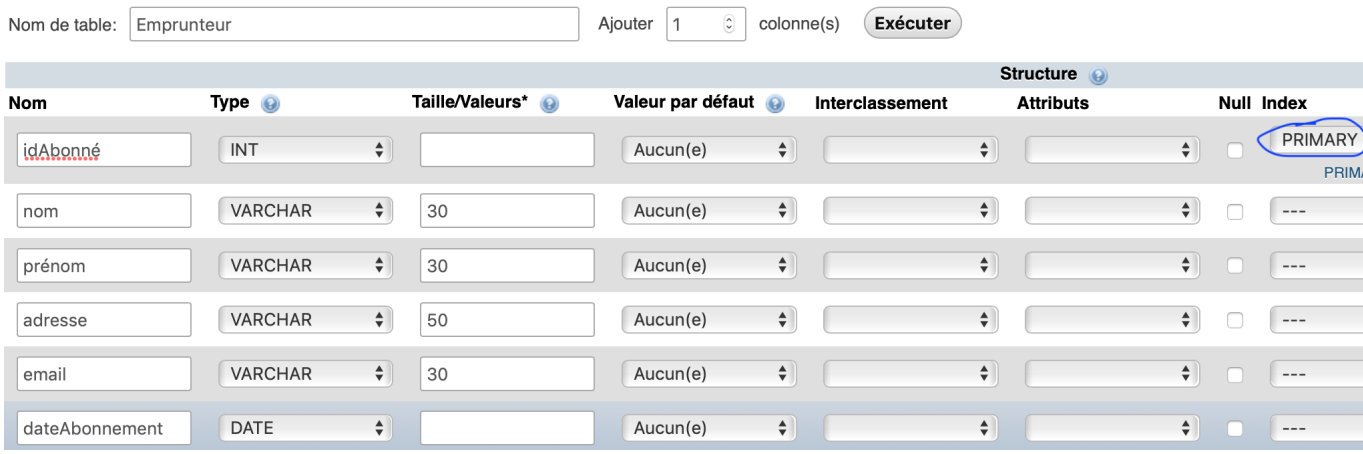
phpMyAdmin est une application graphique permettant de gérer une base de données située sur un serveur (distant ou en local) livrée avec MAMP, LAMP ou WAMP (.....)

On va créer dans l'ordre d'abord les tables sans clé étrangère puis les tables avec clés étrangères

Pour les premières on peut utiliser l'outil graphique qui offre un certain confort (on fait l'économie d'écrire une requête SQL) à condition de s'assurer de **toujours utiliser le même moteur de stockage innoDB** (qui gère les clés étrangères alors que MYISAM ne le gère pas)

(lire ici la documentation mysql `https://dev.mysql.com/doc/refman/8.0/en/create-table-foreign-keys.html`)

Voici une copie d'écran de la création de la table Emprunteur avec phpMyAdmin sur un **serveur distant**



On créé les tables sans clé étrangère avec l'outil graphique c'est à dire les tables Emprunteur, Auteur et Editeur

Ensuite pour les tables avec clés étrangères on va entrer le code SQL dans php-MyAdmin, permettant de créer des tables

```
CREATE TABLE Livre(
codeLivre int(3),
titre varchar(30),
idAuteur int(3),
PRIMARY KEY (codeLivre),
FOREIGN KEY (idAuteur) REFERENCES Auteur (idAuteur)
) ENGINE = InnoDB;
```

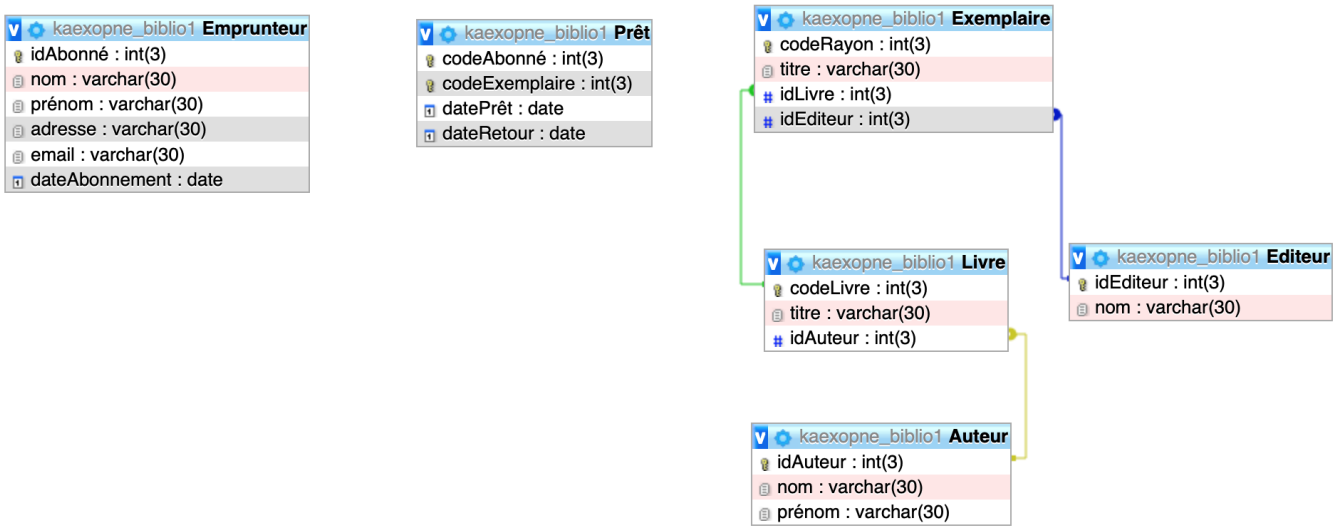
puis on crée avec SQL dans phpMyAdmin la table Exemplaire

```
CREATE TABLE Exemplaire(
codeRayon int(3),
titre varchar(30),
idLivre int(3),
idEditeur int(3),
PRIMARY KEY (codeRayon),
FOREIGN KEY (idLivre) REFERENCES Livre (codeLivre),
FOREIGN KEY (idEditeur) REFERENCES Editeur (idEditeur)
) ENGINE = InnoDB;
```

et finalement la table Prêt :

```
CREATE TABLE Prêt(
codeAbonné int(3),
codeExemplaire int(3),
datePrêt date,
dateRetour date,
PRIMARY KEY (codeAbonné, codeExemplaire),
FOREIGN KEY (codeAbonné) REFERENCES Emprunteur (idAbonné)
FOREIGN KEY (codeExemplaire) REFERENCES Exemplaire (codeRayon)
) ENGINE = InnoDB;
```

Ensuite avec l'outil Concepteur on peut voir les dépendances entre les tables (pour une raison inconnue les liaisons entre Emprunteur et Prêt et Exemplaire n'apparaissent pas)



A ce stade la base de données biblio est créée mais vide !
Il faut maintenant la remplir !

4 Insertion

Ensuite on peut remplir les tables en utilisant l'outil graphique ou en entrant le code SQL dans phpmyAdmin

Par exemple pour entrer des valeurs dans la table Auteur on peut faire :

```
✔ 2 lignes insérées. (traitement en 0.0008 seconde(s).)

INSERT INTO Auteur (idAuteur, nom, prénom) VALUES (3, 'BEAUDELAIRE', 'Charles'), (4, 'THOREAU', 'Henry')
```

et pour la table Auteur

```
INSERT INTO Editeur(idEditeur,nom) VALUES
(1,'Gallimard'),
(2,'Gallmeister'),
(3,'Métailié') ;
```

5 Mise à jour d'une table

1. Imaginons que l'on veuille supprimer les lignes de la table Prêt de l'utilisateur dont le codeAbonné est 235 car il a rendu tous les livres qu'il a emprunté, on utilise la commande SQL suivante

   ```
   DELETE FROM Prêt WHERE codeAbonné = 235
   ```

2. Imaginons que l'abonné dont l'id est 21 a déménagé et a changé d'adresse il habite maintenant 7 rue des orchidées dans ce cas on entre la commande SQL

   ```
   UPDATE Emprunteurs SET adresse = '7 rue des orchidées' WHERE id = 21
   ```

On n'a jamais vu un utilisateur ou un administrateur interroger ou mettre à jour une base de données en faisant directement des requêtes en SQL

C'est pour cela qu'il faut mettre dans le coup un langage de programmation qui guide l'utilisateur par un menu et traduit ses choix en requêtes SQL (Python ou PHP)

6 Python et le module sqlite

Voici les premières lignes de la documentation Python `https://docs.python.org/fr/3/library/sqlite3.html`

> SQLite est une bibliothèque C qui fournit une base de données légère sur disque ne nécessitant pas de processus serveur et qui utilise une variante (non standard) du langage de requête SQL pour accéder aux données. Certaines applications peuvent utiliser SQLite pour le stockage de données internes. Il est également possible de créer une application prototype utilisant SQLite, puis de modifier le code pour utiliser une base de données plus robuste telle que PostgreSQL ou Oracle.

6.1 Création de la base

Voici le code Python pour la création d'une base de données, ici le fichier `biblio.db`

On met les commandes SQL de création dans une constante CREATION de type str puis

1. On crée un objet de type connexion
2. On crée un objet de type Curseur
3. cet objet a une méthode `executescript()`
4. On ferme la connexion

```
import sqlite3

CREATION = """
CREATE TABLE auteur(
idAuteur INTEGER,
nom TEXT,
prenom TEXT,
PRIMARY KEY (idAuteur)
);

CREATE TABLE Emprunteur(
idAbonne INTEGER,
nom TEXT,
prenom TEXT,
adresse TEXT,
email TEXT,
PRIMARY KEY (idAbonne)
);
```

```
CREATE TABLE Editeur(
idEditeur INTEGER,
nom TEXT,
PRIMARY KEY (idEditeur)
);

CREATE TABLE Livre(
codeLivre INTEGER,
titre TEXT,
idAuteur INTEGER,
PRIMARY KEY (codeLivre),
FOREIGN KEY (idAuteur) REFERENCES Auteur (idAuteur)
) ;

CREATE TABLE Exemplaire(
codeRayon INTEGER,
titre TEXT,
idLivre INTEGER,
idEditeur INTEGER,
PRIMARY KEY (codeRayon),
FOREIGN KEY (idLivre) REFERENCES Livre (codeLivre),
FOREIGN KEY (idEditeur) REFERENCES Editeur (idEditeur)
) ;

CREATE TABLE Prêt(
codeAbonné INTEGER,
codeExemplaire INTEGER,
datePrêt  TEXT,
dateRetour TEXT,
PRIMARY KEY (codeAbonné, codeExemplaire),
FOREIGN KEY (codeAbonné) REFERENCES Emprunteur (idAbonné)
FOREIGN KEY (codeExemplaire) REFERENCES Exemplaire (codeRayon)
) ;

"""
#on crée une connexion avec la base de données
connexion = sqlite3.connect('biblio.db')

#on crée une variable de type Cursor
biblio = connexion.cursor()

#on créé les tables
biblio.executescript(CREATION)

#on ferme la connexion
connexion.close()
```

6.2 Insertion des valeurs

Ici il faut distinguer **une première insertion de nombreuses données** (et ici on peut entrevoir la solution d'un transfert de données, sauvegarde d'une base de données dans une autre ou utilisation d'un fichier .csv qui par un programme Python est transformé en une liste de tuples qui ensuite sont insérés dans la base de données (TP)), d'une insertion de quelques éléments peu nombreux, par exemple entrer des nouveaux abonnés à la bibliothèque municipale.

Dans ce dernier cas, par un menu Python ou un formulaire Html un administrateur peut insérer des valeurs dans la table Emprunteur les informations nom, prénom, adresse, email récupérées

```
 import sqlite3

#on crée une connexion avec la base de données
connexion = sqlite3.connect('biblio.db')

#on crée une variable de type Cursor
biblio = connexion.cursor()

#on insère les nouveaux abonnés
while True:
    id = int(input("id ? "))
    nom = input("nom ? ")
    prenom = input("prenom ? ")
    adresse = input("adresse ? ")
    email = input("email ? ")
    biblio.execute('INSERT INTO Emprunteur VALUES (?,?,?,?,?)',
     (id,nom,prenom,adresse,email))
    choix = input('On continue ? (O/n)')
    if choix == "n":
        break


#on valide les modifications
connexion.commit()
#on ferme la connexion
connexion.close()
```

6.3 Modification de certains attributs

L'administrateur veut modifier les adresses de certains abonnés qui ont déménagé

```
import sqlite3

#on crée une connexion avec la base de données
connexion = sqlite3.connect('biblio.db')
```

```
#on crée une variable de type Cursor
biblio = connexion.cursor()

#on modifie les adresses d'abonnés
while True:
    id = int(input("id ? "))
    adresse = input("nouvelle adresse ? ")

    biblio.execute('UPDATE Emprunteur SET adresse = ? WHERE idAbonne = ?
     (adresse,id))
    choix = input('On continue ? (O/n)')
    if choix == "n":
        break


#on valide les modifications
connexion.commit()
#on ferme la connexion
connexion.close()
```

6.4 Suppression de certaines lignes

L'administrateur veut supprimer les prêts de tel utilisateur qui a rendu un certain nombre d'exemplaires

```
import sqlite3

#on crée une connexion avec la base de données
connexion = sqlite3.connect('biblio.db')

#on crée une variable de type Cursor
biblio = connexion.cursor()

#on supprime des emprunts
while True:
    idAbonne = int(input("idAbonné ? "))
    idExemplaire = int(input("idExemplaire ? "))

    biblio.execute('DELETE FROM Prêt WHERE codeAbonné = ? and codeExempl
     (idAbonne,idExemplaire))
    choix = input('On continue ? (O/n)')
    if choix == "n":
        break


#on valide les modifications
connexion.commit()
```

```
#on ferme la connexion
connexion.close()
```

6.5 Requêtes d'interrogation

Un utilisateur veut interroger la base de données pour avoir tous les livres (pas les exemplaires) de la bibliothèque d'un auteur donné

La requête SQL (jointure) pour cela est :

```
SELECT a.nom, l.titre
FROM Auteur a, Livre l
WHERE a.idAuteur = l.idAuteur and a.nom = 'CHAR'
```

SQLITE n'exprime pas les jointures comme SQL

```
import sqlite3

#on crée une connexion avec la base de données
connexion = sqlite3.connect('biblio.db')
connexion.row_factory = sqlite3.Row

#on crée une variable de type Cursor
biblio = connexion.cursor()


#on visualise tous les livres d'un Auteur
biblio.execute("SELECT nom, titre FROM Livre INNER JOIN Auteur ON
Auteur.idAuteur =  Livre.idAuteur WHERE nom = 'CHAR'")
for row in biblio:
    print('{} : {}'.format(row['nom'],row['titre']))

#on ferme la connexion
connexion.close()
```

A partir de la base de données Messagerie dont le schéma relationnel est donné ci-dessous

Contact (**idContact**, nom, prénom, email)
Message (**idMessage**, contenu, dateEnvoi, *idEmetteur*, *idPrédecesseur*)
Envoi (**idMessage**, **idContact**,)

Avec Python et SQLITE :

1. Créer la base de données
2. Insérer des valeurs dans les tables
3. Consulter la base