TP 1 — Premiers pas avec les bases de données

Dans ce TP, vous allez vous initier au langage SQL et utiliser `DBeaver` pour apprendre à :

- créer une base de donnée et une table ;
- spécifier le type des données ;
- modifier la structure d'une table.

1 Mise en place

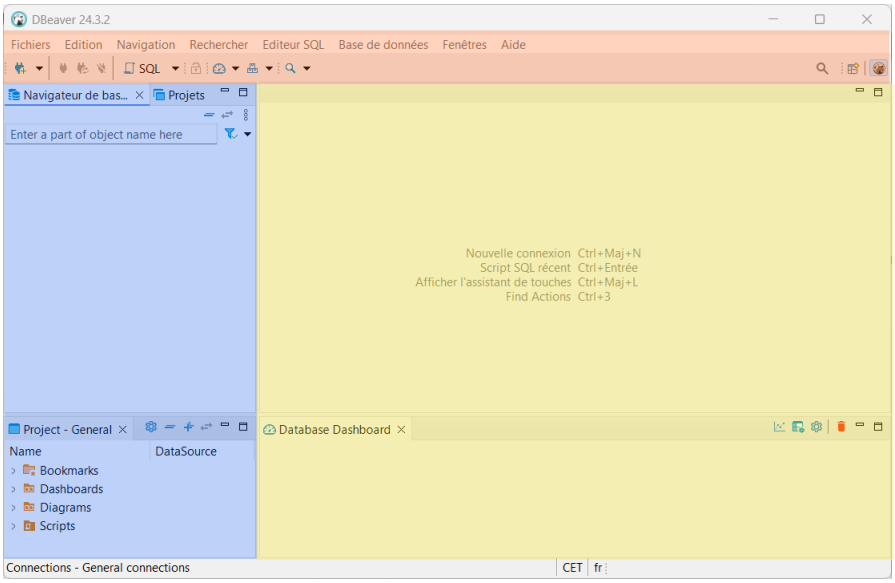
1) Placez-vous dans votre répertoire `Documents` et créez à l'intérieur les sous-dossiers `R105_BDD` ▸ `TP1`.

2) Ouvrez `DBeaver`, et fermez les éventuelles bases de données ouvertes.

3) Ouvrez la boite de dialogue Nouvelle connexion, choisissez SQLite puis Suivant.

4) Cliquez sur Create... et placez-vous dans le dossier `Documents` ▸ `BUT_INFO` ▸ `R105_BDD` ▸ `TP1` que vous venez de créer. Puis nommez le fichier `tp1_<nomDeFamille>.sqlite`.

Vous devez obtenir cette fenêtre :



La partie en rouge contient les habituels menus et raccourcis. À gauche (en bleu sur l'image), c'est la partie principale qui permet de manipuler la base de données et d'écrire des requêtes.

Et à gauche, en bleu sur l'image, diverses informations relatives à la base de données. Sous l'onglet Navigateur de bases de données, observez que noter base de contient pour le moment aucune table. Ouvrez l'éditeur SQL via le menu SQL » Script SQL. C'est ici que vous écrirez vos lignes de codes.

À rendre

Au fur et à mesure du TP, vous allez écrire des instructions en SQL (ces instructions s'appellent *des requêtes*). Il faudra enregistrer votre travail dans le fichier source `tp1_<nomDeFamille>.sql`. Vous enregistrerez votre base de données avec le nom `tp1_<nomDeFamille>.sqlite` (attention à ne pas confondre le fichier de la base de données et le code de vos requêtes SQL, cf. TP 1).

Le TP contient également des questions : vous devez rédiger un compte-rendu et le rendre *obligatoirement au format PDF* avec vos réponses. Ce compte-rendu devra être nommé `tp1_compteRendu_<nomDeFamille>.pdf`.

À la fin du TP, assemblez dans une ZIP les fichiers `tp1_<nomDeFamille>.sqlite`, `tp1_<nomDeFamille>.sql` et `tp1_compteRendu_<nomDeFamille>.pdf`. Vous nommerez l'archive `tp1_<nomDeFamille>.zip` et vous la déposerez sur EPREL.

2 Un peu de théorie

2.1 Création de table et types de données

Placez-vous dans l'éditeur et copiez le code suivant :

tp1_<nomDeFamille>.sql

```
1  CREATE TABLE IF NOT EXISTS etudiants (
2      etudiant_id INTEGER(8),
3      etudiant_nom VARCHAR(55),
4      etudiant_prenom VARCHAR(55),
5      etudiant_telephone INTEGER(10),
6      etudiant_age INTEGER(2),
7      etudiant_note DECIMAL(3, 1)
8  );
9
10 SELECT * FROM etudiants;
```

Ce code crée une *table* dans la base de données, nommée `etudiants` (au pluriel). Cette table se comporte comme un tableau, les colonnes étant les attributs `etudiant_id`, `etudiant_nom` (au singulier), etc.

La ligne 10 demande à SQLite d'afficher toutes les lignes présentes dans la table `etudiants` ; mais comme cetta table est vide, rien ne s'affiche.

Les attributs de la table ont un *type*, qui décrit la nature des données contenues. Il existe de très nombreux types en SQL : nombres (entiers ou décimaux), chaines de caractères, dates, durées, booléens, etc. Les nombres entre parenthèses indiquent les tailles de chaque attribut. Par exemple, `etudiant_prenom VARCHAR(55)` indique que le prénom est une chaine de caractères de taille au plus 55.

Les types d'attributs en SQL

Les types de données en SQL sont *standards*. En revanche, les moteurs de bases de données, i.e. les SGBDD n'implémentent pas tous les types de données, et ne les implémentent pas de la même manière.

La liste des types disponibles en SQLite est accessible sur la documentation : Datatypes in SQLite. Le moteur SQLite contient en réalité un nombre *assez restreint* de types possibles, car c'est un SGBDD minimaliste. MySQL, par exemple, gère beaucoup plus de types de données, cf. cette page.

SQLite est capable de convertir automatiquement les types du standard SQL vers les types qu'il implémente. Un attribut déclaré en `VARCHAR(55)` sera automatiquement typé en `TEXT`.

Afin de prendre les bonnes habitudes dès maintenant, *il est obligatoire* d'utiliser les types du standard SQL.

2.2 Valeurs par défaut

Lorsque l'on ajoute des lignes dans une table (cf. TP 2), il peut être utile d'attribuer à certains champs une valeur par défaut, comme suit :

```SQL
CREATE TABLE IF NOT EXISTS comediens (
    comedien_id INTEGER(6),
    comedien_nom VARCHAR(60),
    lieu_naissance VARCHAR(60) DEFAULT 'Inconnu'
);
```

Lorsque l'on ajoutera des valeurs dans la table `comediens`, si l'attribut `lieu_naissance` n'est pas renseigné, il prendra automatiquement la valeur `'Inconnu'`. Notez la présence des apostrophes qui délimitent les chaines de caractères. Observez aussi que la requête se termine par un point-virgule. À l'inverse, on peut souhaiter qu'une valeur soit explicitement laissée vide si elle n'est pas renseignée : c'est le mot-clé `NULL` qui le permet.

```SQL
CREATE TABLE IF NOT EXISTS comediens (
    comedien_id INTEGER(6),
    comedien_nom VARCHAR(60),
    lieu_naissance VARCHAR(60) DEFAULT NULL
);
```

Si l'on souhaite *interdire* qu'une valeur soit non renseignée, on utilise `NOT NULL` :

```SQL
CREATE TABLE IF NOT EXISTS comediens (
    comedien_id INTEGER(6),
    comedien_nom VARCHAR(60) NOT NULL, -- L'attribut comedien_nom sera obligatoire.
    lieu_naissance VARCHAR(60) DEFAULT NULL
);
```

2.3 Les clés primaires

Dans la table `comediens`, l'attribut `comedien_id` doit être unique et obligatoirement renseigné. Il permet donc d'identifier chaque ligne de manière unique. Deux comédien·nes qui auraient même nom et même lieu de naissances ne pourraient être différencié·es que par leur identifiant.

On dit que l'attribut `comedien_id` est la *clé primaire* de la table `comediens`. Cette valeur ne peut pas être `NULL` et deux lignes distinctes prennent des valeurs disctinctes. Une clé primaire est identifiée par le mot-clé `PRIMARY KEY` :

SQL
```sql
CREATE TABLE IF NOT EXISTS comediens (
    comedien_id INTEGER NOT NULL PRIMARY KEY, -- Note : la taille n'est plus précisée.
    comedien_nom VARCHAR(60) NOT NULL,
    lieu_naissance VARCHAR(60) DEFAULT NULL
);
```

2.4 Commentaires

Dans vos codes SQL, il est possible d'ajouter des commentaires, qui seront ignorées lors de l'exécution, mais permettent de fournir des explications à l'intérieur du code. Le bon usage des commentaires est une compétence essentielle en programmation.

Il existe des commentaires sur une seule ligne (cf. ligne 2 de l'exemple précédent), qui commencent par deux tirets *`-- Le commentaire commence ici`*. On peut aussi écrire des commentaires sur plusieurs lignes *`/*entre ces deux symboles */`* :

SQL
```sql
CREATE TABLE IF NOT EXISTS comediens ( -- Création de la table comediens
    comedien_id INTEGER(6) NOT NULL,
    /* comedien_nom VARCHAR(60),
    lieu_naissance VARCHAR(60) DEFAULT NULL */
);
```

Cela créera une table… avec une seule colonne, pas très intéressante.

3 Mise en pratique : modélisation du BUT informatique

On se propose de modéliser à l'aide d'une base de données le BUT informatique (de manière simplifiée). Nous allons créer (ou modifier) des tables pour représenter les étudiant·es, les enseignant·es et les resssources.

Exercice 1 (La table `etudiants`)**.** On reprend la table `etudiants` présentée dans la section 2.1. Nous allons modifier la structure de la table.

1) Supprimer l'attribut `etudiant_note`, il n'est pas utile. Pour ce faire, exécutez la requête :

tp1_<nomDeFamille>.sql
```sql
ALTER TABLE etudiants
DROP COLUMN etudiant_note;
```

2) L'attribut `etudiant_age` devrait être facultatif et `NULL` par défaut. En SQLite, il n'est pas possible de modifier *a posteriori* les propriétés d'une colonne. Écrivez donc une requête qui supprime cette colonne, puis la recrée en imposant la valeur `NULL` par défaut. Consultez la documentation pour trouver comment ajouter une colonne dans une table.

3) Nous souhaitons ajouter un attribut indiquant l'année d'étude de l'étudiant·e : les valeurs possibles sont des entiers (1, 2 ou 3). Quel sera le type de la colonne ? Écrire la requête correspondante. Utiliser le mot-clé `CHECK`, en consultant la documentation.

4) Enfin, chaque étudiant·e est dans un groupe (A, B ou C). Ajoutez la colonne nécessaire à votre table.

5) Vérification — Votre table doit comporter les colonnes suivantes :

`etudiant_id` la clé primaire.
`etudiant_nom` type chaine de caractères de taille 55.
`etudiant_prenom` type chaine de caractères de taille 55.
`etudiant_telephone` type entier de taille 10.
`etudiant_age` type entier, `NULL` par défaut.
`etudiant_annee` de valeur 1, 2 ou 3.
`etudiant_groupe` de valeur A, B ou C.

Exercice 2 (La table `enseignants`)**.** Un·e enseignant·e est décrit·e par quatre informations : un identifiant (nombre entier) unique, son prénom, son nom et son nombre d'heures de cours par an.

1) Quels sont les types de chaque colonne ? Quelle sera la clé primaire ?

2) Écrire une requête SQL qui crée la table `enseignants` avec les bonnes colonnes. Nommez les colonnes `enseignant_id`, `enseignant_prenom`, `enseignant_nom` et `enseignant_heures`. Tous les attributs sont obligatoires donc ne peuvent pas prendre la valeur `NULL`.

Exercice 3 (La table `ressources`)**.** Nous allons maintenant créer la table qui représentera les ressources. Chaque ressource est caractérisée trois éléments.

- Son code (R1.02, R1.13, R1.05, etc.).
- Son libellé complet.
- L'enseignant·e en charge de la ressource.

1) Quels sont les types des deux premières colonnes ? Quelle est la clé primaire ?

Pour le troisième attribut, nous voudrions pouvoir utiliser les données présentes dans la table `enseignants`. Dans cette table, chaque ligne est identifiable de manière unique grâce à l'attribut `enseignant_id`. Cela permet de faire la liaison entre les deux tables.

2) Compte tenu de l'indication précédente, quel sera le type de la colonne qui correpond à l'enseignant·e en charge de la ressource ?

3) Écrire une requête SQL qui crée la table `ressources`, en donnant aux colonnes les noms `ressource_id`, `ressource_libelle`, `enseignant_id`.

La notion de clé étrangère

Dans la table `ressources`, on dit que l'attribut `enseignant_id` est la *clé étrangère* de la table `enseignants`.

Le concept de clé étrangère est crucial pour comprendre les bases de données, car il permet de relier plusieurs tables entre elles. Dans les modèles de données, les liaisons entre les tables correspondent très souvent à des clés étrangères. Ce thème va être étudié de façon plus approfondie dans un prochain TP.

4 Le mot de la fin

À la fin de ce TP, vous devez savoir :

- écrire une requête qui crée une table dans une base de données (`CREATE TABLE`), en précisant les types de chaque colonne
- ajouter ou supprimer une colonne dans une table ;
- attribuer à certains attributss une valeur par défaut, ou interdire la valeur `NULL` ;
- indiquer qu'une colonne est une clé primaire ;
- choisir pour chaque colonne d'une table, le bon type de données.

Quelques bonnes pratiques

- Par convention, les mots-clés de SQL s'écrivent en lettres majuscules.
- Les noms de tables sont généralement au pluriel, puisque chaque table contiendra plusieurs lignes (souvent, un très grand nombre !).
- Les noms des colonnes dans une table s'écrivent au singulier, car ils décrivent les attributs de *chaque ligne*.
- Si une table se nomme `elements` (au pluriel), on peut nommer les colonnes de cette table en suivant la convention `element_<nomAttribut>` (au singuler). Cela alourdit les noms, mais améliore la lisibilité et peut se révéler utile lorsque les bases de données deviennent complexes.