Bases de données relationnelles en SQL

Implémentation de contraintes supplémentaires

Lorsque l'on crée une relation, on peut ajouter des contraintes d'intégrités autres que clés primaires (`PRIMARY KEY`) ou étrangères (`FOREIGN KEY ... REFERENCES ...`).

- `UNIQUE` : la valeur de l'attribut doit être unique
- `NOT NULL` : la valeur de l'attribut de peut pas être laissée vide (attention, contrairement à Python la chaîne de caractère vide `''` n'est pas égale à `NULL`!)(j'insiste : `''` $\neq$ `NULL`)

Exemple d'utilisation de `UNIQUE` et `NOT NULL`

```
CREATE TABLE livre (titre TEXT NOT NULL,
                    editeur TEXT NOT NULL,
                    annee INTEGER NOT NULL,
                    isbn TEXT UNIQUE NOT NULL PRIMARY KEY);
```

Remarque : il est redondant d'écrire `PRIMARY KEY` et `NOT NULL UNIQUE`. Cependant, à cause d'un bug de DBeaver avec SQLITE on prendra la peine d'ajouter explicitement `NOT NULL UNIQUE` pour un attribut de contrainte clé primaire.

Pour les chaînes de caractères, il existe est possible d'expliciter un nombre fixe de caractères ou un nombre variable :

- `VARCHAR(`n`)` : chaîne de caractère de taille variable d'au plus n caractères
- `CHAR(`n`)` : chaîne de caractère de taille n (si la valeur est inférieure à n, des espaces sont ajoutées)

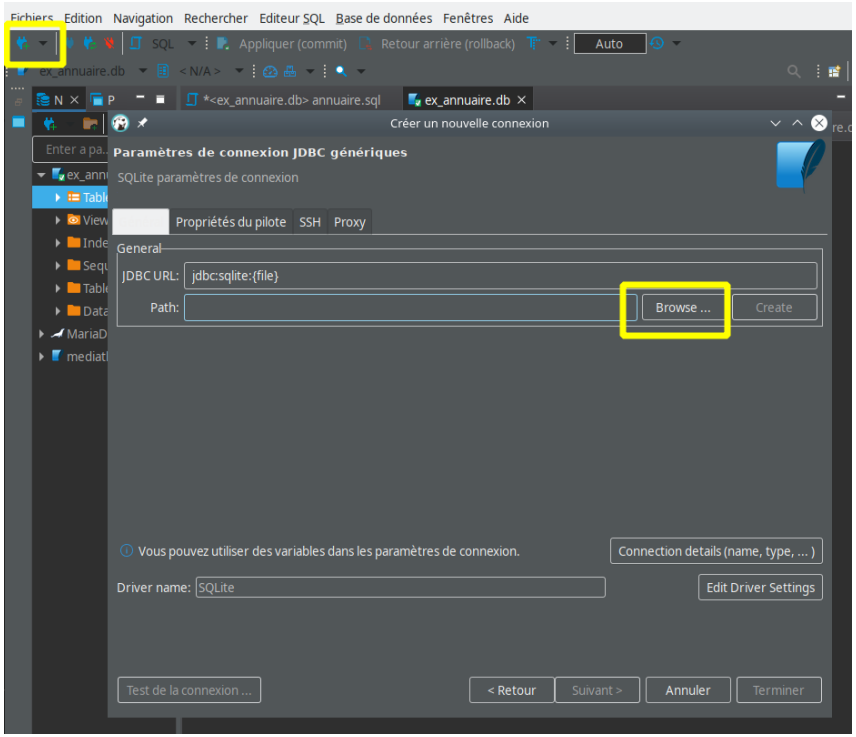
Par exemple, on peut explicitement indiquer que l'`isbn` possède 14 caractères et que `titre` et `editeur` du livre ne dépassent pas 90 caractères de la façon suivante :

```
CREATE TABLE livre (titre VARCHAR(90) NOT NULL,
                    editeur VARCHAR(90) NOT NULL,
                    annee INTEGER NOT NULL,
                    isbn CHAR(14) UNIQUE NOT NULL PRIMARY KEY);
```

Activité avec DBeaver

Vous allez créer votre première base avec DBeaver CE.

1. Créer un fichier vide nommé `annuaire.db` (par exemple dans un terminal, dans le répertoire de votre choix écrire : `touch annuaire.db`).
2. Ouvrir DBeaver CE
3. Créer une nouvelle connexion :
    1. de type SQLite
    2. avec `Browse...`, sélectionner le fichier créé `annuaire.db`.
    3. puis `Terminer`



Maintenant que la base est créer, vous pouvez effectuer des requêtes. Vous allez les écrire dans un fichier où vous pourrez les exécuter toutes d'un seul coup (peu recommandé), ou ligne par ligne. Comme un notebook, chaque ligne est un block que vous pouvez exécuter 0, 1 ou plusieurs fois.

1. `Éditeur SQL > Nouveau script SQL`
2. écrire par exemple `CREATE TABLE test (id INT);`
3. puis positionner le prompt sur la ligne et exécuter l'instruction SQL (clic sur le triangle de lecture ou `CTRL + Entrée`)
4. Sauf erreur, la relation `test` est créée. Pour en être sûr, réaliser à nouveau l'étape précédente et normalement il y aura une erreur puisque la relation *existe déjà*.

Pour finir avec cette petite prise en main, vous allez insérer des données et afficher le contenu de la relation :

8. Dans une autre ligne du fichier, écrire `INSERT INTO test VALUES (1);` et exécuter plusieurs fois cette ligne (pour insérer plusieurs fois l'entité (1) dans la relation $test$).
9. Puis écrire et exécuter `SELECT * FROM test;`.

Si tout s'affiche comme prévu, nous allons voir une dernière fonctionnalité : afficher le diagramme de la base puis effacer la relation…

10. Dans le panneau de gauche de `annuaire.db`, il doit être possible de déplier la base pour voir s'afficher en dessous `Tables ; Views ; Indexes ; Sequences ; Table Triggers ; Data Types`.
11. Double-cliquer sur `Tables`
12. Les propriétés des différentes tables (table = *relation*) s'affichent dans l'onglet `Propriétés`
13. Actualiser la base avec `F5` puis
14. Cliquer sur l'onglet `ER Diagram` pour voir le diagramme de la relation.
15. Si tout va bien, vous avez quelque chose qui ressemble à cela.

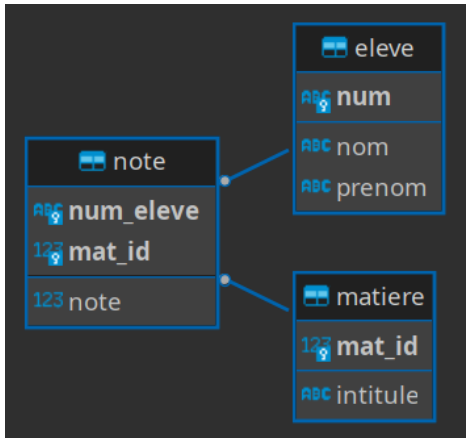
- `FOREIGN KEY(auteur_id) REFERENCES auteur(a_id)` ou `auteur_id INT REFERENCES auteur(a_id)`
- `CHECK (0 <= note AND note <= 20)`
- `CHECK (ville <> 'Paris')` : pour $\neq$

Types

- `TEXT` : texte de longueur variable
- `VARCHAR(`n`)` : texte de longueur variable inférieure à n
- `CHAR(`n`)` : texte de longueur n
- `INT` ou `INTEGER` : nombres entiers
- `DECIMAL(`c`,`d`)` : nombre flottant de c chiffres dont d pour la partie décimale
- `DATE` : une date au format `AAAA-MM-JJ`
- `TIME` : une heure au format `hh:mm:ss`
- `TIMESTAMP` : un instant (date et heure) au format `AAAA-MM-JJ hh:mm:ss`

Tables

- `CREATE TABLE nom_table (...)`
- `DROP TABLE nom_table`
- `DROP TABLE IF EXISTS nom_table`
- `INSERT INTO nom_table VALUES (...), (...), ..., ();`
- `INSERT INTO nom_table(attr1, attr2) VALUES ...`
-

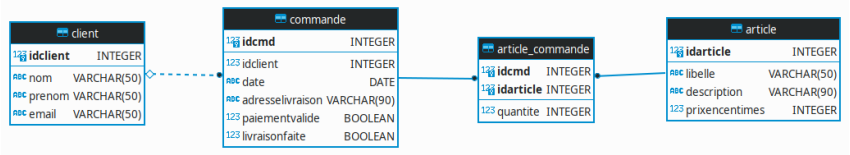


Suppression

1. Dans quel ordre faut-il effacer les tables ? Pourquoi ?
2. Tester dans DBeaver.

Activité - base de donnée `vente.db`

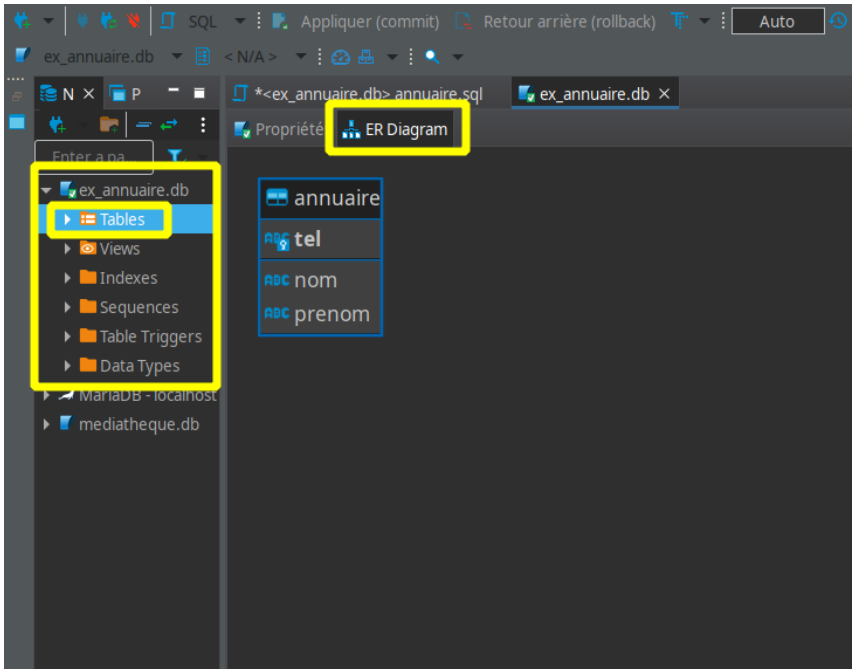
Créer la base de donnée `vente.db` respectant les contraintes suivantes :



Résumés et notes

Contraintes

- `PRIMARY KEY(auteur_id)` ou `auteur_id INT PRIMARY KEY`
- `PRIMARY KEY(auteur_id, isbn)`



1. Pour finir, vous allez supprimer la relation `test` de la base en exécutant la requête `DROP TABLE test;`.

Activité - retour sur `annuaire.db`

Créer dans la base de donnée `annuaire.db` l'unique relation suivante :

$$annuaire(\underline{tel} : String, nom : String, prenom : String)$$

Contraintes d'intégrités :

- *tel* : clé primaire, non nulle, unique, chaîne de caractère de taille variable (100 caracères max)

- *nom* : non nul, chaîne de caractère de taille variable (100 max)
- *prenom* : non nul , chaîne de caractère de taille variable (20 max)

Implémentation des contraintes de domaines et nombres décimaux

En SQL, on peut ajouter des **contraintes de domaines**. Par exemple on peut imposer un attribut `age` d'être toujours positif ou nul.

Pour cela, on utilise l'instruction `CHECK`. Ce qui donne :

- `CHECK (age >= 0)`

On peut utiliser les opérateurs logiques `NOT`, `AND` et `OR` ainsi que les parenthèses pour les priorités.

Pour les nombre flottants, il existe un type qui prend en charge le nombre de chiffres après la virgule et s'occupe des arrondis le cas échéant.

- `DECIMAL(`c`,`d`)` : permet d'afficher un nombre de c chiffres (en tout) dont d sont après la virgule

Par exemple pour afficher un nombre compris entre 0 et 999 avec une précision de 3 décimales, on écrira

```
nombre DECIMAL(6, 3)
```

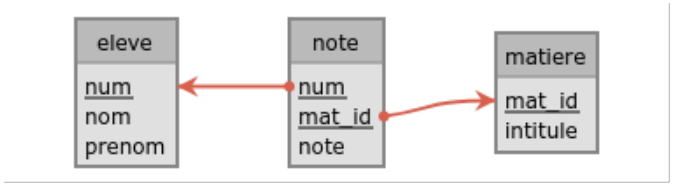
car il y a 6 chiffres en tout dont 3 pour la partie décimale.

Activité - base de donnée `note.db`

Création

Créer la base de donnée `note.db` et les trois relations suivantes :

- $eleve(nom : String, prenom : String, \underline{num} : String)$
- $matiere(intitule : String, \underline{mat\_id} : Int)$
- $note(\underline{num\_eleve} : String, \underline{mat\_id} : Int, note : Float)$



Attention à respecter les contraintes suivantes :

- *eleve* :
  - *nom* : taille variable (taille 100 max), non nul
  - *prenom* : taille variable (max 100), non nul
  - *num* : taille variable (max 20), clé primaire (unique, non nul)
- *matiere* :
  - *intitule* : taille variable (max 100), non nul
  - mat_id : entier, clé primaire (unique non nul)
- *note* :
  - *num_eleve* : taille variable (max 20), reférence à $eleve(num)$, clé primaire
  - *mat_id* : entier, référence à $matiere(mat\_id)$, clé primaire
  - *note* : nombre décimal de 4 chiffres dont 2 après la virgule, non nul, supérieure ou égale à 0, inférieure ou égal à 20

Résultat attendu :