Créer une base SQLite avec DB Browser

Construire de A à Z la base d'une école — classes, élèves, professeurs, cours et points — entièrement à la souris dans DB Browser : tables, types, contraintes, clés étrangères et saisie des données, avec le piège des clés étrangères désactivées par défaut.

5TTR 6TTR Découverte

On passe à la pratique : tu vas construire la base d'une école, avec ses classes, ses élèves, ses professeurs, ses cours et ses points. Tout se fait à la souris, dans DB Browser. C'est cette base qui servira aux cours suivants.

🎯 Objectifs

À la fin de ce cours, tu seras capable de :

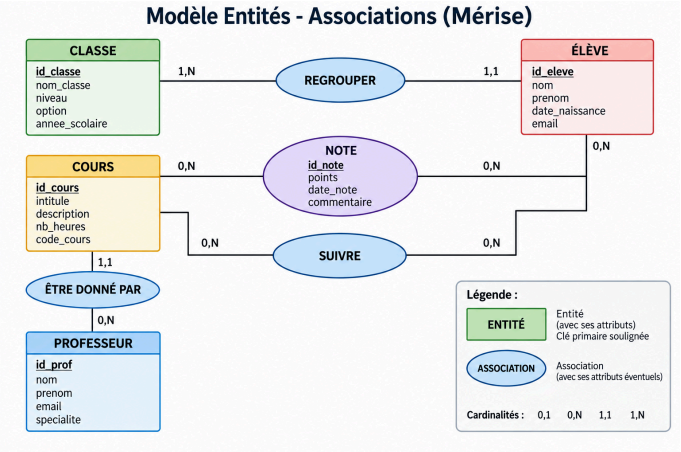
1. **Créer** une base SQLite et ses tables avec l'interface de DB Browser.
2. **Choisir** le bon type de stockage pour une donnée, y compris une date.
3. **Poser** les contraintes utiles : clé primaire, `NOT NULL`, `UNIQUE`, `CHECK`, valeur par défaut.
4. **Déclarer** une clé étrangère et **activer** leur vérification — qui ne l'est pas par défaut.
5. **Reconnaître** une relation N-N et la modéliser par une table d'association.
6. **Saisir** des données et comprendre dans quel ordre le faire.

ℹ️ Ce cours se fait entièrement dans l'interface graphique. DB Browser affiche au passage le code SQL qu'il exécute pour toi : tu apprendras à l'écrire toi-même dans un cours ultérieur. Ici, tu le **LIS**, tu ne le tapes pas.

Le schéma à construire

Avant de cliquer, on modélise. Voici ce que la base doit représenter :

- une **classe** regroupe des **élèves** ;
- un **cours** est donné par un **professeur** ;
- un **élève** suit plusieurs cours, un cours est suivi par plusieurs élèves ;
- chaque élève reçoit des **points** dans les cours qu'il suit.



Traduit en tables :

Table	Rôle	Relation
`classe`	3TTR, 4TTR, 5TTR, 6TTR	—
`professeur`	Les enseignants	—
`eleve`	Les élèves	`classe_id` → `classe`
`cours`	Les cours donnés	`professeur_id` → `professeur`
`inscription`	Quel élève suit quel cours	**table d'association** N-N
`note`	Les points d'un élève dans un cours	→ `eleve` et → `cours`

🧠 **POURQUOI `INSCRIPTION` EXISTE.** Un élève suit plusieurs cours *et* un cours a plusieurs élèves : c'est une relation **N-N**. Aucune des deux tables ne peut porter la référence de l'autre — il en faudrait plusieurs par ligne. On crée donc une troisième table dont chaque ligne représente **un couple** : un élève, un cours.

Créer le fichier

1. Lance DB Browser.
2. **Nouvelle base de données**. Choisis l'emplacement et nomme le fichier `ecole.db`.
3. Une fenêtre *Créer une table* s'ouvre aussitôt : **ferme-la**, on va d'abord régler un point important.

Activer les clés étrangères — à faire en premier

Onglet **Éditer les pragmas**, coche `Foreign Keys`, puis *Enregistrer*.

⚠️ **LE PIÈGE N°1 DE SQLITE.** Les clés étrangères sont **désactivées par défaut**, pour des raisons de compatibilité avec d'anciennes versions. Sans ce réglage, tu peux déclarer toutes les clés étrangères que tu veux : SQLite les enregistre mais **ne les vérifie pas**. Tu pourras inscrire un élève dans une classe qui n'existe pas, et t'en apercevoir six mois plus tard.

Ce réglage vaut **pour la session en cours**, pas pour le fichier lui-même. Chaque logiciel qui ouvre la base doit le refaire — y compris tes futurs programmes.

Créer une table

Onglet **Structure de la base** → **Créer une table**. Nomme-la `classe`, puis ajoute les champs avec le bouton **Ajouter** :

Nom	Type	Cases à cocher
`id`	INTEGER	**PK** (clé primaire)
`nom`	TEXT	**NN** (non nul), **U** (unique)
`annee`	INTEGER	**NN**
`local`	TEXT	—

En bas de la fenêtre, DB Browser affiche en direct le code qu'il va exécuter :

```
CREATE TABLE classe (
    id      INTEGER PRIMARY KEY,
    nom     TEXT    NOT NULL UNIQUE,
```

```
    annee   INTEGER NOT NULL,
    local   TEXT
);
```

Lis-le à chaque table que tu crées. Tu n'as pas à l'écrire aujourd'hui, mais faire le lien entre les cases que tu coches et le texte qui apparaît est le meilleur moyen d'apprendre la syntaxe sans effort.

> 💡 `ID` **EN INTEGER + PK : LE CAS PARTICULIER À CONNAÎTRE.** En SQLite, une colonne déclarée exactement `INTEGER PRIMARY KEY` se remplit **toute seule** — tu peux laisser la case vide à la saisie, le numéro est attribué automatiquement. Inutile donc de cocher **AI** (auto-incrément) : cette option existe, mais elle a un coût et ne sert que dans un cas rare, empêcher la réutilisation d'un identifiant supprimé. **Par défaut, ne la coche pas.**

Les types disponibles

SQLite ne connaît que **cinq classes de stockage** :

Classe	Contenu	Exemple
`NULL`	L'absence de valeur	un élève sans classe
`INTEGER`	Nombre entier	`17`, `-3`
`REAL`	Nombre à virgule	`13.5`
`TEXT`	Chaîne de caractères	`Dupont`
`BLOB`	Données binaires brutes	une photo

La liste déroulante de DB Browser en propose d'autres (`VARCHAR`, `DATETIME`, `BOOLEAN` …) : SQLite les accepte, mais les ramène à l'une de ces cinq classes. Autant choisir directement celles qui existent vraiment.

Trois absences à connaître :

Ce qui manque	Ce qu'on utilise à la place
Type booléen	`INTEGER` : `0` = faux, `1` = vrai
Type date / heure	`TEXT` au format ISO `2026-09-07`
Décimal exact (monnaie)	`REAL` (attention aux arrondis) ou un `INTEGER` de centimes

> 💡 **POURQUOI LE FORMAT** `AAAA-MM-JJ` **ET PAS** `07/09/2026` **?** Parce qu'un tri alphabétique sur du texte ISO donne exactement le tri chronologique : `2026-09-07` vient bien après `2026-08-31`, dans les deux ordres. Avec `07/09/2026`, le tri serait absurde. C'est aussi le seul format que comprennent les fonctions de date de SQLite.

Les contraintes

Dans la fenêtre de création, chaque champ dispose de cases à cocher et d'une colonne pour les conditions :

Contrainte	Où	Effet	Exemple
Clé primaire	case **PK**	Identifie la ligne	`id`
Non nul	case **NN**	Interdit l'absence de valeur	`nom` d'un élève
Unique	case **U**	Interdit les doublons	`email` d'un professeur
Vérification	champ **Check**	Impose une condition	`valeur BETWEEN 0 AND 20`
Valeur par défaut	champ **Défaut**	Valeur si rien n'est saisi	`1` pour `periode`
Clé étrangère	champ **Clé étrangère**	Lien vers une autre table	`classe(id)`

🧠 **UNE CONTRAINTE N'EST PAS UNE PUNITION : C'EST UNE GARANTIE.** Chaque contrainte que tu poses est une erreur que la base refusera à ta place, pour toujours. Une vérification `valeur BETWEEN 0 AND 20` sur les points, c'est la certitude qu'aucun programme, aucun import, aucune faute de frappe ne créera un 200/20.

Les clés étrangères

Déclarer le lien

Crée la table `eleve` avec les champs `id`, `nom`, `prenom`, `date_naissance` et `classe_id`. Pour `classe_id` (type INTEGER), clique dans la colonne **Clé étrangère** et choisis la table `classe`, puis la colonne `id`.

```sql
CREATE TABLE eleve (
    id              INTEGER PRIMARY KEY,
    nom             TEXT NOT NULL,
    prenom          TEXT NOT NULL,
    date_naissance  TEXT,
    classe_id       INTEGER REFERENCES classe(id)
);
```

Remarque que `classe_id` **n'est pas** coché *non nul* : c'est volontaire. Un élève qui vient d'arriver n'est pas encore affecté à une classe. Une clé étrangère vide signifie « pas de lien » — et c'est une information valable.

Vérifier que la protection fonctionne

Va dans *Parcourir les données*, table `eleve`, insère une ligne et tape `999` dans `classe_id` — un identifiant de classe qui n'existe pas. Puis **écris les modifications** (`Ctrl+S`).

- **Si tu obtiens une erreur** `FOREIGN KEY constraint failed` → parfait, la protection est active.
- **Si la ligne est acceptée** → le pragma n'est pas activé. Retourne le cocher, puis supprime la ligne fautive.

⚠️ Fais ce test **UNE FOIS PAR BASE** que tu crées. C'est dix secondes, et ça évite de découvrir des données incohérentes bien plus tard.

La table d'association

Pour `inscription`, la particularité est que la clé primaire porte sur **deux colonnes** à la fois : coche **PK** sur `eleve_id` **et** sur `cours_id`. Le même élève ne pourra alors pas être inscrit deux fois au même cours.

```sql
CREATE TABLE inscription (
    eleve_id INTEGER NOT NULL REFERENCES eleve(id),
    cours_id INTEGER NOT NULL REFERENCES cours(id),
    PRIMARY KEY (eleve_id, cours_id)
);
```

Ici, les deux colonnes sont **non nulles** : une inscription sans élève ou sans cours n'a aucun sens.

Saisir les données

Dans la grille

Onglet **Parcourir les données**, choisis la table dans la liste déroulante, puis le bouton **Insérer un nouvel enregistrement**. Tape les valeurs directement dans la grille, cellule par cellule.

Laisse la colonne `id` vide : SQLite la remplit à l'enregistrement.

Pour les clés étrangères, tu dois saisir l'**identifiant** de la ligne liée — le `1` de la classe 3TTR, pas le texte `3TTR`. D'où l'intérêt de commencer par les tables sans dépendances, dont tu noteras les identifiants au fur et à mesure.

> ⚠️ **N'OUBLIE PAS D'ÉCRIRE LES MODIFICATIONS** (`Ctrl+S`). Tant que le bouton *Écrire les modifications* est actif, ton travail n'est que dans la mémoire de DB Browser. Si le logiciel se ferme, tout est perdu.

L'ordre de saisie compte

Avec les clés étrangères actives, on ne peut pas inscrire un élève dans une classe qui n'existe pas encore. L'ordre est donc imposé par les dépendances :

```
classe, professeur   (ne dépendent de rien)
      ↓
eleve, cours         (dépendent des précédentes)
      ↓
inscription, note    (dépendent de eleve et cours)
```

C'est la même logique en sens inverse pour les suppressions : on ne peut pas supprimer une classe qui contient encore des élèves. La base t'oblige à rester cohérent — c'est exactement son rôle.

> 💡 **ET POUR SAISIR 200 LIGNES ?** Personne ne le fait à la souris. On utilise alors soit un import de fichier (*Fichier → Importer → Table depuis un fichier CSV*), soit des instructions SQL — c'est ce que contient le fichier `ecole.sql` fourni. Tu apprendras à les écrire dans un cours ultérieur.

La base fournie

Tu trouves dans la sidebar deux fichiers :

- `ecole.db` — la base complète, prête à ouvrir : 4 classes, 8 professeurs, 11 cours, 33 élèves, 207 inscriptions et 367 points.
- `ecole.sql` — le script qui reconstruit cette base à zéro, pour la restaurer si besoin (*Fichier → Importer → Base de données depuis un fichier SQL*).

Elle contient volontairement trois situations qui te serviront dans les cours suivants :

Situation	Où	Ce qu'elle permet d'apprendre
Un élève sans classe	`Camille Nouveau` , `classe_id` vide	Le comportement des jointures face aux valeurs absentes
Un cours sans aucun inscrit	`Néerlandais`	La différence entre `JOIN` et `LEFT JOIN`
Des classes d'effectifs différents	10, 9, 7 et 6 élèves	Des regroupements aux résultats parlants

> 💡 **GARDE UNE COPIE INTACTE DU FICHIER.** Tu vas modifier cette base pendant les exercices. Duplique `ecole.db` en `ecole-original.db` maintenant : restaurer, ce sera juste une copie de fichier. C'est l'un des conforts de SQLite.

🧪 Exercices

Tout se fait dans l'interface de DB Browser. Tu peux utiliser l'onglet *Exécuter le SQL* pour **vérifier** tes résultats avec un `SELECT`, mais la création et la saisie se font à la souris.

Exercice 1 — Reconstruire la structure ★★☆☆☆

Crée une base vide `mon-ecole.db` et construis toi-même les six tables à l'interface, sans regarder la base fournie.

Compare ensuite avec `ecole.db` (onglet *Structure de la base*, qui affiche le SQL de chaque table). Ta version n'a pas à être identique — mais pour chaque écart, demande-toi lequel des deux choix est le meilleur, et pourquoi.

Exercice 2 — Poser les bonnes contraintes ★★☆☆☆

Pour chaque colonne, indique les contraintes que tu poserais et **justifie** :

Colonne	Tes contraintes
`professeur.email`	
`note.valeur`	
`cours.periodes`	
`eleve.date_naissance`	
`classe.nom`	
`note.periode` (1 ou 2)	

Applique-les ensuite à ta base `mon-ecole.db`.

Exercice 3 — Faire échouer ses propres contraintes ★★☆☆☆

Dans `mon-ecole.db`, essaie de saisir dans la grille **six** lignes qui doivent toutes être refusées — une par contrainte de l'exercice 2. Par exemple : un point à 25, deux classes du même nom, un cours sans professeur.

Pour chaque tentative, note le **message d'erreur exact** affiché par DB Browser au moment d'écrire les modifications.

> Une contrainte qu'on n'a jamais vue se déclencher est une contrainte dont on n'est pas sûr.

Exercice 4 — La date piégée ★★★☆☆

Dans la table `eleve` de `mon-ecole.db`, saisis deux élèves de test avec ces dates de naissance :

- `2008-03-14`
- `14/03/2008`

a) Les deux valeurs sont-elles acceptées ? Pourquoi, d'après ce que tu sais du typage de SQLite ? b) Dans *Parcourir les données*, clique sur l'en-tête de la colonne `date_naissance` pour trier. L'ordre obtenu est-il correct ? c) Vérifie avec `SELECT prenom, date_naissance, strftime('%Y', date_naissance) FROM eleve;` — que renvoie la fonction pour chacune des deux lignes ? d) Quelle contrainte ajouterais-tu à la table pour empêcher le mauvais format ? e) Supprime ensuite ces deux lignes de test.

Exercice 5 — Étendre le schéma ★★★☆☆

L'école veut enregistrer les **absences** : pour un élève, une date, une demi-journée (matin ou après-midi), et si l'absence est justifiée ou non.

a) Crée la table `absence` à l'interface, avec ses contraintes et sa clé étrangère. b) Comment représentes-tu « justifiée ou non » ? Justifie ton choix de type. c) Comment empêches-tu d'enregistrer deux fois la même absence, pour le même élève, le même jour, la même demi-journée ? d) Saisis trois absences de test, puis vérifie que ta protection se déclenche bien à la quatrième. e) Vérifie ton contenu avec `SELECT * FROM absence;`.

✍️ À retenir

- **Les clés étrangères sont désactivées par défaut** : à activer dans les pragmas, pour chaque logiciel qui ouvre la base.
- SQLite n'a que **cinq classes de stockage** : `NULL`, `INTEGER`, `REAL`, `TEXT`, `BLOB`.
- **Pas de type date** : on stocke du texte ISO `AAAA-MM-JJ`, parce que son tri alphabétique est chronologique.
- **Pas de booléen** : `0` et `1` dans un `INTEGER`.
- Une colonne `INTEGER` en clé primaire **se remplit toute seule**.
- Une relation **N-N** se modélise par une table d'association dont la clé primaire porte sur les deux colonnes.
- Une clé étrangère **peut** être vide : ça veut dire « pas de lien ».
- L'ordre de saisie suit les dépendances ; l'ordre de suppression les remonte.
- Dans DB Browser, rien n'est enregistré tant qu'on n'a pas **écrit les modifications**.
- Le SQL affiché en bas de la fenêtre de création est le même que celui que tu écriras plus tard : prends l'habitude de le lire.

Suite

La base est prête et remplie. Passe au cours Interroger une base SQLite : SELECT et COUNT pour en tirer des réponses : qui est dans quelle classe, combien d'élèves par cours, quelle moyenne par classe.