SQLite

SQLite range une base de données complète — tables, relations, contraintes — dans un seul fichier local, sans rien à installer ni à démarrer. Ce cours explique ce que ça permet, où on le rencontre déjà, et surtout où sont les limites.

5TTR 6TTR Découverte

Une base de données complète — des tables, des relations, des contraintes, du SQL — rangée dans **un seul fichier** que tu peux copier sur une clé USB. Pas de logiciel serveur à démarrer, pas de compte, pas de mot de passe. C'est ça, SQLite.

🎯 Objectifs

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

1. **Expliquer** ce qu'est une base de données stockée dans un fichier local.
2. **Citer** trois usages où SQLite est le bon choix, et trois où il ne l'est pas.
3. **Nommer** les cinq limitations principales de SQLite et dire pourquoi chacune existe.
4. **Reconnaître** le typage dynamique et ce qu'il autorise.
5. **Installer** DB Browser for SQLite et ouvrir une base existante.

Une base de données dans un fichier

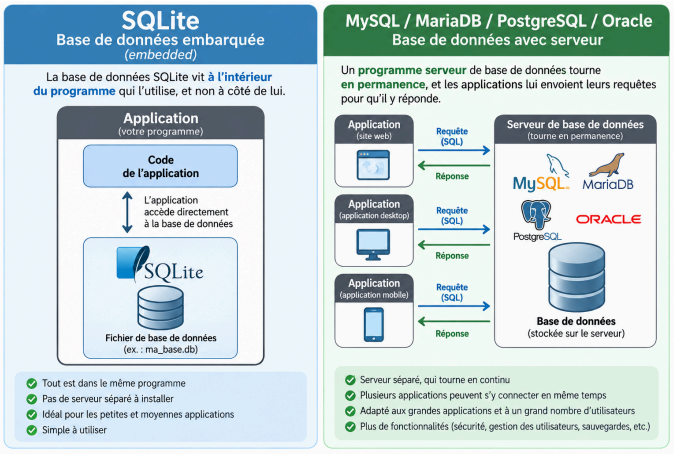
Le principe

SQLite n'est pas un programme qu'on lance : c'est une **bibliothèque**, c'est-à-dire du code que ton application embarque. Quand ton programme veut lire ou écrire une donnée, il appelle une fonction, et cette fonction lit ou écrit dans un fichier.

```
ton programme (Python, une appli mobile, DB Browser…)
   │
   │  appel de fonction — rien ne circule sur un réseau
   ▼
bibliothèque SQLite
   │
   ▼
ecole.db          ← UN fichier. C'est toute la base de données.
```

Ce fichier contient **tout** : les tables, les données, les index, les contraintes. Tu peux le copier sur une clé USB, l'envoyer par mail, le déposer dans un dépôt Git. Sur un autre ordinateur, il s'ouvre à l'identique — aucune installation, aucune restauration, aucun import.

Le format de ce fichier est **stable depuis 2004**, et ses auteurs se sont engagés à le maintenir lisible jusqu'en 2050 au moins. C'est l'une des raisons pour lesquelles des bibliothèques et des services d'archives l'utilisent comme format de conservation à long terme.



🧠 **RETIENS L'EXPRESSION.** On dit que SQLite est une base de données **embarquée** (*embedded*) : elle vit à l'intérieur du programme qui l'utilise, et non à côté de lui.

L'autre façon de faire : le serveur

SQLite n'est pas la seule approche. La plupart des autres systèmes de bases de données — MySQL, MariaDB, PostgreSQL, Oracle — fonctionnent autrement : un **programme serveur** tourne en permanence, et les applications lui **envoient** leurs requêtes pour qu'il y réponde.

```
application  ──requête via le réseau──►  serveur de base de données
                                                    │
                                                    ▼
                                            fichiers de données
```

Ce serveur écoute sur le réseau, vérifie des mots de passe, gère les droits de chaque utilisateur et arbitre entre les applications qui veulent écrire en même temps. C'est plus lourd à installer et à administrer, et c'est ce qui rend possible qu'un grand nombre de personnes travaillent simultanément sur les mêmes données.

ℹ️ **SELON TON PARCOURS, CE PARAGRAPHE EST UN RAPPEL OU UN APERÇU.**

Si tu as déjà travaillé avec MySQL, HeidiSQL ou phpMyAdmin, tu reconnais le modèle : il fallait démarrer un serveur, créer un utilisateur, se connecter. SQLite supprime toutes ces étapes.

Si tu ne l'as pas encore vu, retiens seulement ceci : **il existe des bases de données qui vivent dans un fichier, et d'autres qui vivent dans un programme serveur qu'on interroge à distance.** Tu rencontreras le second modèle plus tard ; le SQL que tu apprends ici te servira dans les deux cas.

Aucune des deux approches n'est meilleure dans l'absolu : elles répondent à des besoins différents. La fin de ce cours te donne les critères pour choisir.

Où SQLite se trouve déjà

D'après ses auteurs, SQLite est **le moteur de base de données le plus déployé au monde**. Ça surprend, jusqu'à ce qu'on regarde où il se trouve :

Où	Ce qu'il y stocke
Android et iOS	Contacts, SMS, historique d'appels, réglages de chaque application
Firefox, Chrome, Safari	Historique, favoris, cookies, mots de passe enregistrés
Windows 10 et 11	Plusieurs composants système et applications intégrées
Applications de bureau	Logiciels de photo, de comptabilité, jeux, lecteurs multimédias
Systèmes embarqués	Avionique, automobile, télévisions, appareils médicaux

Le point commun : ce sont des logiciels **installés chez l'utilisateur**. Ils ont besoin de stocker des données structurées, mais on ne peut évidemment pas demander à chaque personne qui installe un jeu ou un navigateur d'administrer un serveur de base de données au passage.

> 💡 **UN ORDRE DE GRANDEUR.** Ton téléphone contient probablement plusieurs centaines de fichiers SQLite : chaque application en a au moins un.

Les limitations

C'est la partie la plus importante de ce cours. Un technicien qui connaît les limites d'un outil sait **quand ne pas l'utiliser** — c'est exactement ce qu'on attend de toi.

Un seul écrivain à la fois

C'est **la** limitation structurante. Quand un programme écrit dans la base, SQLite verrouille le fichier : personne d'autre ne peut écrire pendant ce temps.

- **Lire à plusieurs :** aucun problème, des dizaines de programmes peuvent lire simultanément.
- **Écrire à plusieurs :** impossible. Les autres attendent leur tour, et au-delà d'un certain délai ils reçoivent l'erreur `database is locked`.

Pourquoi ? Parce qu'il n'y a **personne pour arbitrer**. SQLite n'a que les verrous du système de fichiers pour se coordonner, là où un serveur de base de données dispose d'un programme dédié qui met les écritures en file d'attente.

> ⚠️ **CONSÉQUENCE CONCRÈTE.** Un site de vente en ligne où 200 personnes commandent en même temps : SQLite n'est pas le bon outil. Un site de consultation lu par 200 personnes qui n'écrivent rien : SQLite tient très bien.

Pas d'utilisateurs, pas de droits

SQLite ne connaît ni comptes, ni mots de passe, ni privilèges : il n'y a pas de `CREATE USER`, pas de `GRANT`. Ces notions n'ont pas de sens pour une bibliothèque qui ouvre un fichier.

La sécurité d'une base SQLite, c'est donc la sécurité **du fichier**. Qui peut le lire lit toutes les données ; qui peut l'écrire modifie tout. Les seules protections disponibles sont les permissions du système de fichiers.

> ⚠️ **NE METS JAMAIS UN FICHIER `.db` DANS UN DOSSIER SERVI PAR UN SERVEUR WEB.** S'il est accessible par une URL, n'importe qui le télécharge et l'ouvre chez lui. C'est une fuite de données complète, en un clic.

Pas d'accès réseau

Le fichier doit être sur le disque de la machine qui l'utilise. SQLite ne sait pas se connecter à distance.

Et la tentation classique — poser le `.db` sur un partage réseau pour que plusieurs postes l'utilisent — est **explicitement déconseillée par ses auteurs** : les mécanismes de verrouillage de fichiers des partages réseau (SMB, NFS) ne sont pas fiables. Le risque n'est pas la lenteur, c'est la **base corrompue**.

Le typage est dynamique

Dans la plupart des systèmes de bases de données, une colonne déclarée « nombre entier » refuse le texte. Pas ici :

```
CREATE TABLE test (age INTEGER);
INSERT INTO test VALUES ('bonjour');   -- accepté !
```

SQLite applique une **affinité de type** (*type affinity*) : il *essaie* de convertir la valeur vers le type déclaré et, s'il n'y arrive pas, il la stocke telle quelle. Le type est porté par la **valeur**, pas par la colonne.

C'est souple, et c'est un piège : une erreur de saisie qui aurait été bloquée ailleurs passe inaperçue, jusqu'au jour où un calcul donne un résultat aberrant.

> 💡 **LA PARADE.** Depuis SQLite 3.37, une table peut être déclarée stricte : elle refuse alors les valeurs du mauvais type.
>
> ```
> CREATE TABLE test (age INTEGER) STRICT;
> ```

`ALTER TABLE` très limité

Modifier une table existante est beaucoup plus restreint qu'ailleurs. On peut renommer une table, ajouter une colonne, renommer une colonne, en supprimer une — **mais pas changer le type d'une colonne, ni ajouter une contrainte** à une table déjà créée.

Quand il le faut vraiment, la méthode officielle consiste à créer une nouvelle table avec la bonne structure, y copier les données, supprimer l'ancienne et renommer la nouvelle. D'où une règle pratique : **avec SQLite, on soigne le schéma dès le départ**.

Et aussi

Absent de SQLite	Pourquoi ça compte
Procédures stockées	Toute la logique doit vivre dans l'application

Absent de SQLite	Pourquoi ça compte
Réplication, sauvegarde à chaud native	Sauvegarder = copier le fichier, base fermée
`RIGHT JOIN` et `FULL OUTER JOIN` avant la version 3.39	Vérifie ta version avant de t'en servir
Réglages de mémoire, de cache, de connexions	Rien à administrer — ce qui est aussi son intérêt

En revanche, ce que beaucoup croient être une limite n'en est pas une : **la taille**. Une base SQLite peut techniquement atteindre plusieurs centaines de téraoctets. Ce n'est jamais le volume qui disqualifie SQLite, c'est la concurrence en écriture.

Choisir SQLite, ou pas

Situation	Choix	Pourquoi
Application de bureau ou mobile pour un seul utilisateur	**SQLite**	Rien à installer chez l'utilisateur
Prototype, projet scolaire, jeu de données à analyser	**SQLite**	Zéro configuration, le fichier se partage
Site web de consultation (blog, catalogue, documentation)	**SQLite**	Beaucoup de lectures, peu d'écritures
Format d'échange ou d'archivage de données structurées	**SQLite**	Format stable, lisible dans 20 ans
Site marchand, réservation, plusieurs personnes qui écrivent	**Serveur**	Écritures simultanées
Données utilisées depuis plusieurs machines	**Serveur**	SQLite n'est pas réseau
Besoin de droits par utilisateur, d'audit, de réplication	**Serveur**	Fonctions absentes de SQLite

> 🧠 **LA BONNE QUESTION N'EST PAS « LEQUEL EST LE PLUS PUISSANT ? »** mais « **combien de personnes écrivent en même temps, et depuis combien de machines ? »**. Si la réponse est « une seule », SQLite fait le travail avec un fichier au lieu d'une infrastructure à administrer.

L'outil du cours : DB Browser for SQLite

Un fichier `.db` n'est pas lisible dans un éditeur de texte : c'est un format binaire. Il faut un logiciel pour l'ouvrir — ce sera **DB Browser for SQLite**, libre et gratuit.

Installation :

1. Va sur sqlitebrowser.org (lien aussi dans la sidebar).
2. Télécharge la version **Windows 64 bits (.msi)**, *standard installer*.
3. Installe-le et lance-le.

Ce que tu y trouves, en onglets :

Onglet	À quoi il sert
Structure de la base	Voir et modifier les tables, colonnes, index, contraintes
Parcourir les données	Consulter et éditer le contenu, table par table
Éditer les pragmas	Régler le comportement du moteur — dont `foreign_keys`
Exécuter le SQL	Écrire et lancer des requêtes

> ⚠️ **LE PIÈGE N°1 DE DB BROWSER.** Tes modifications ne sont **pas** enregistrées dans le fichier tant que tu n'as pas cliqué sur « **Écrire les modifications** » (`Ctrl+S`). Un bouton *Annuler les modifications* juste à côté annule tout ton travail. Prends le réflexe d'écrire régulièrement.

🧪 Exercices

Exercice 1 — Fichier ou serveur ? ★☆☆☆☆

Pour chaque situation, dis si tu choisis SQLite ou une base sur serveur, et justifie **en une phrase** :

#	Situation
a	Une application de gestion de collection de films, installée sur le PC d'un particulier
b	Le site de réservation en ligne d'un cinéma, 300 places par séance
c	Un programme qui analyse 50 000 lignes de relevés météo
d	Le logiciel de caisse d'un magasin, avec trois caisses simultanées reliées au même stock
e	Une application mobile de suivi d'entraînement sportif, sans compte utilisateur
f	L'intranet d'une école : 40 professeurs encodent les points en même temps le 20 juin

Exercice 2 — Le typage dynamique ★★☆☆☆

Dans DB Browser, onglet *Exécuter le SQL* :

```sql
CREATE TABLE essai (id INTEGER PRIMARY KEY, age INTEGER);
INSERT INTO essai (age) VALUES (17), ('dix-sept'), ('18');
SELECT id, age, typeof(age) FROM essai;
```

a) Quel type SQLite a-t-il attribué à chacune des trois valeurs ? b) Pourquoi `'18'` n'est-il pas traité comme `'dix-sept'` ? c) Que renvoie `SELECT SUM(age) FROM essai;` ? Le résultat te paraît-il fiable ? d) Recommence en créant la table avec `STRICT`. Que se passe-t-il à l'insertion ?

Exercice 3 — Chercher dans la documentation officielle ★★☆☆☆

Sur `sqlite.org`, trouve et note :

a) La taille maximale théorique d'une base SQLite. b) Le nombre maximal de colonnes par table, par défaut. c) Ce que la documentation répond à la question : *« SQLite peut-il être utilisé sur un partage réseau ? »*

Pour chaque réponse, cite la page utilisée. La documentation est en anglais : c'est l'occasion de travailler la lecture technique.

Exercice 4 — Explorer une vraie base ★★☆☆☆

Firefox range tes favoris et ton historique dans un fichier SQLite nommé `places.sqlite`.

a) Retrouve-le sur ton poste (dossier de profil Firefox). Note sa taille. b) **Copie-le ailleurs**, puis ouvre la copie dans DB Browser — jamais l'original, et Firefox fermé. c) Combien de tables contient-il ? Note les trois qui te paraissent les plus importantes. d) Dans l'onglet *Parcourir les données*, trouve la table qui contient les URL visitées. e) En quoi cet exercice illustre-t-il la limitation « pas d'utilisateurs, pas de droits » ?

Exercice 5 — Argumenter face à un client ★★★☆☆

Un client te dit :

> « Mon informaticien m'a dit que SQLite, c'est un jouet, que ce n'est pas une vraie base de données. Vous en pensez quoi ? »

Rédige ta réponse en cinq lignes maximum : ni complaisance, ni jargon. Elle doit contenir un fait vérifiable, une nuance, et une question en retour pour cadrer son besoin réel.

✍️ À retenir

- SQLite range **toute une base de données dans un fichier local**, copiable et partageable tel quel.
- C'est une **bibliothèque embarquée** dans le programme : rien à démarrer, aucun compte, aucun réseau.
- D'autres systèmes fonctionnent avec un **programme serveur** qu'on interroge à distance : plus lourd, mais fait pour le travail simultané.
- C'est le moteur le plus déployé au monde, parce qu'il est dans les téléphones, les navigateurs et les applications de bureau.
- **Un seul écrivain à la fois** : c'est la limite qui décide, bien plus que la taille des données.
- **Aucune gestion d'utilisateurs** : la sécurité, ce sont les permissions du fichier.
- **Pas d'accès réseau** — et poser un `.db` sur un partage réseau expose à la corruption.
- Le **typage est dynamique** : une colonne `INTEGER` accepte du texte, sauf en mode `STRICT`.
- `ALTER TABLE` **est limité** : on soigne le schéma dès le départ.
- La bonne question : *combien de personnes écrivent en même temps, depuis combien de machines ?*

Suite

Passe au cours Créer une base SQLite avec DB Browser : tu vas construire la base `ecole.db` — classes, élèves, professeurs, cours et points — et découvrir au passage le piège des clés étrangères désactivées par défaut.