Ajouter des lignes : INSERT

La table des professeurs existe mais elle est vide. Tu y ajoutes des lignes en SQL, une par une puis plusieurs à la fois, tu observes ce que la base refuse, et tu importes un fichier CSV.

4TTR 5TTR 6TTR Découverte

Six professeurs doivent entrer dans la base de l'école. Tu en encodes un à la main, puis trois d'un coup. Les deux derniers arrivent dans un fichier envoyé par le secrétariat. À chaque étape, la base vérifie ce que tu lui donnes, et elle n'hésite pas à refuser.

> 💡 **FICHIERS DE DÉPART**
>
> Télécharge dans la barre latérale `mon-ecole-eleves.db` (la table `eleve` avec 12 élèves) et `nouveaux-professeurs.csv` (le fichier du secrétariat). Ouvre la base dans DB Browser for SQLite. Le SQL s'écrit dans l'onglet **Exécuter le SQL**.

🎯 Objectifs

À la fin de cet article, tu seras capable de :

1. **Ajouter** une ligne avec `INSERT INTO … VALUES`, en nommant les colonnes.
2. **Prévoir** ce que la base écrit dans les colonnes que tu ne remplis pas.
3. **Ajouter** plusieurs lignes en une seule instruction.
4. **Reconnaître**, à la lecture du message, quelle contrainte a refusé une ligne.
5. **Importer** un fichier CSV dans une table, puis dans une autre avec `INSERT … SELECT`.
6. **Enregistrer** tes ajouts dans le fichier de la base.

Préparer la table professeur

La base téléchargée ne contient pas encore de table pour les professeurs. Exécute ce script. Il ne crée la table que si elle n'existe pas encore : `IF NOT EXISTS` veut dire *si elle n'existe pas*.

```sql
CREATE TABLE IF NOT EXISTS professeur (
    id        INTEGER PRIMARY KEY,
    nom       TEXT    NOT NULL,
    prenom    TEXT    NOT NULL,
    email     TEXT    NOT NULL UNIQUE,
    actif     INTEGER NOT NULL DEFAULT 1,
    telephone TEXT
);
```

En deux lignes, ce que disent ces règles. `INTEGER PRIMARY KEY` donne à chaque professeur un numéro unique, que SQLite attribue tout seul. `NOT NULL` rend une colonne obligatoire, `UNIQUE` interdit deux fois la même adresse, et `DEFAULT 1` écrit `1` dans `actif` quand on ne précise rien.

Vérifie que la table est vide :

```sql
SELECT * FROM professeur;
```

```
id | nom | prenom | email | actif | telephone
---+-----+--------+-------+-------+----------
```

> ⚠️ Si la table existait déjà et contient des lignes, retélécharge `mon-ecole-eleves.db` : les résultats de cet article supposent une table vide au départ.

Ajouter une ligne

Sophie Dubois est la première à entrer dans la base. Elle a donné son numéro de téléphone.

```sql
INSERT INTO professeur (nom, prenom, email, telephone)
VALUES ('Dubois', 'Sophie', 's.dubois@ecole.be', '0471 23 45 67');
```

L'instruction se lit presque comme une phrase. `INSERT INTO` veut dire *insère dans*, et `VALUES` veut dire *les valeurs* : « insère dans la table `professeur`, dans les colonnes `nom`, `prenom`, `email` et `telephone`, les valeurs `'Dubois'`, `'Sophie'` … ».

> 📖 **NOUVELLE NOTION : INSERT**
>
> L'instruction `INSERT INTO table (colonnes) VALUES (valeurs)` ajoute une ligne dans une table. Les valeurs sont données dans le même ordre que les colonnes : la première valeur va dans la première colonne citée, et ainsi de suite. Les textes s'écrivent entre apostrophes.
>
> En anglais : GB ***insert***, *insérer*.

Vérifie tout de suite le résultat :

```sql
SELECT * FROM professeur;
```

```
id | nom    | prenom | email             | actif | telephone
---+--------+--------+-------------------+-------+--------------
1  | Dubois | Sophie | s.dubois@ecole.be | 1     | 0471 23 45 67
```

Deux colonnes sont remplies alors que l'`INSERT` ne les cite pas :

- `id` vaut `1` : SQLite a choisi le numéro lui-même, grâce à `INTEGER PRIMARY KEY` ;
- `actif` vaut `1` : c'est la valeur prévue par `DEFAULT 1`.

> 💡 **UN `SELECT` APRÈS CHAQUE AJOUT.** DB Browser annonce que l'instruction s'est bien exécutée, mais seul un `SELECT` montre ce que la table contient vraiment. Prends l'habitude de vérifier à chaque fois.

Ajouter plusieurs lignes d'un coup

Trois autres professeurs arrivent : Ludovic Lambrechts, Marc Vermeulen et Anne Leroy. Aucun n'a donné de téléphone. Plutôt que d'écrire trois `INSERT`, on sépare les groupes de valeurs par des virgules :

```sql
INSERT INTO professeur (nom, prenom, email)
VALUES
    ('Lambrechts', 'Ludovic', 'l.lambrechts@ecole.be'),
    ('Vermeulen',  'Marc',    'm.vermeulen@ecole.be'),
    ('Leroy',      'Anne',    'a.leroy@ecole.be');
```

Sous la zone de requête, DB Browser indique le nombre de lignes ajoutées : trois.

```sql
SELECT * FROM professeur;
```

```
id | nom        | prenom  | email                | actif | telephone
---+------------+---------+----------------------+-------+--------------
1  | Dubois     | Sophie  | s.dubois@ecole.be    | 1     | 0471 23 45 67
2  | Lambrechts | Ludovic | l.lambrechts@ecole.be | 1     | NULL
3  | Vermeulen  | Marc    | m.vermeulen@ecole.be  | 1     | NULL
4  | Leroy      | Anne    | a.leroy@ecole.be      | 1     | NULL
```

Cette fois, `telephone` n'était pas citée et n'a pas de valeur par défaut : elle reçoit `NULL`, qui veut dire « aucune valeur ».

> 🧠 **UNE COLONNE QUE TU NE CITES PAS DANS L'`INSERT` REÇOIT SA VALEUR PAR DÉFAUT SI ELLE EN A UNE, SINON `NULL`.** La clé primaire `INTEGER PRIMARY KEY`, elle, reçoit le numéro suivant.

Toujours nommer les colonnes

On peut écrire un `INSERT` sans la liste des colonnes. Essaie :

```sql
INSERT INTO professeur VALUES ('Lemaire', 'Claire', 'c.lemaire@ecole.be');
```

```
table professeur has 6 columns but 3 values were supplied
```

Le message dit : « la table professeur a 6 colonnes, mais 3 valeurs ont été fournies ». Sans liste, SQLite exige une valeur pour **chaque** colonne, dans l'ordre exact de la table, `id` compris.

> ⚠️ **L'`INSERT` SANS LISTE DE COLONNES EST DANGEREUX.** Si tu inverses le nom et le prénom, SQLite ne voit rien : ce sont deux textes. Et le jour où quelqu'un ajoute une colonne à la table, tous tes anciens `INSERT` cessent de fonctionner. Avec la liste des colonnes, chaque valeur sait où elle va.

Quand la base dit non

Les contraintes de la table ne sont pas décoratives. Provoque-les.

Un prénom qui manque

Claire Lemaire arrive, mais le prénom a été oublié dans l'instruction :

```sql
INSERT INTO professeur (nom, email)
VALUES ('Lemaire', 'c.lemaire@ecole.be');
```

```
NOT NULL constraint failed: professeur.prenom
```

Une adresse déjà utilisée

Un nouveau professeur, Pierre Dubois, reçoit par erreur l'adresse de Sophie Dubois :

```sql
INSERT INTO professeur (nom, prenom, email)
VALUES ('Dubois', 'Pierre', 's.dubois@ecole.be');
```

```
UNIQUE constraint failed: professeur.email
```

Ce que la base a gardé

```sql
SELECT * FROM professeur;
```

La table contient toujours les quatre mêmes professeurs. Une ligne refusée n'est pas ajoutée, même en partie.

Lire un message d'erreur

Les messages sont en anglais, mais ils suivent toujours le même modèle : **la contrainte**, puis `constraint failed` (« contrainte non respectée »), puis **la table et la colonne** en cause.

Message	Ce qu'il veut dire	Que faire
`NOT NULL constraint failed: professeur.prenom`	la colonne `prenom` est obligatoire et tu ne l'as pas remplie	ajouter la valeur manc
`UNIQUE constraint failed: professeur.email`	cette adresse existe déjà dans la table	vérifier la valeur, ou ch
`table professeur has 6 columns but 3 values were supplied`	pas de liste de colonnes, et pas assez de valeurs	nommer les colonnes
`no such column: Lemaire`	un texte écrit sans apostrophes est pris pour un nom de colonne	entourer le texte d'apc

> 🧠 **LA BASE NE DIT PAS NON PAR MÉCHANCETÉ.** Chaque refus protège les données contre une erreur : un professeur sans prénom, deux comptes avec la même adresse. Le message indique précisément quelle règle a joué et sur quelle colonne. Lis-le avant de modifier ton instruction au hasard.

Importer un fichier CSV

Le secrétariat envoie les deux derniers professeurs dans un fichier `nouveaux-professeurs.csv`. Ouvre-le avec le Bloc-notes :

```
nom;prenom;email
Martin;Julie;j.martin@ecole.be
Renard;Paul;p.renard@ecole.be
```

> 📖 **NOUVELLE NOTION : LE FICHIER CSV**
>
> Un **fichier CSV** est un fichier texte qui contient un tableau : une ligne du fichier par ligne du tableau, avec les valeurs séparées par un caractère fixe (souvent la virgule ou le point-virgule). La première ligne donne souvent le nom des colonnes. Les tableurs et les bases de données savent tous le lire et l'écrire.
>
> En anglais : GB ***Comma-Separated Values***, des *valeurs séparées par des virgules*.

Ce fichier n'a pas les mêmes colonnes que la table : ni `id`, ni `actif`, ni `telephone`. On procède donc en deux temps : on importe le fichier dans une **table de passage**, puis on copie ses lignes dans `professeur`.

Du fichier vers une table de passage

1. Menu **Fichier > Importer > Table depuis un fichier CSV…**, puis choisis `nouveaux-professeurs.csv`.
2. Comme nom de table, écris `import_prof`.
3. Coche l'option qui indique que la première ligne contient les noms de colonnes.
4. Choisis le point-virgule `;` comme séparateur.
5. Vérifie l'aperçu : trois colonnes `nom`, `prenom`, `email` et deux lignes. Valide.

```sql
SELECT * FROM import_prof;
```

```
nom    | prenom | email
-------+--------+------------------
Martin | Julie  | j.martin@ecole.be
Renard | Paul   | p.renard@ecole.be
```

De la table de passage vers professeur

Au lieu de `VALUES`, on donne à l'`INSERT` le résultat d'un `SELECT` :

```sql
INSERT INTO professeur (nom, prenom, email)
SELECT nom, prenom, email
FROM import_prof;
```

Cela se lit : « insère dans `professeur`, dans les colonnes `nom`, `prenom`, `email`, les lignes que renvoie ce `SELECT` ».

> 📖 **NOUVELLE NOTION : INSERT … SELECT**
>
> `INSERT INTO table (colonnes) SELECT …` ajoute dans une table toutes les lignes renvoyées par un `SELECT`. Le `SELECT` doit renvoyer autant de colonnes que la liste en cite, dans le même ordre. C'est la façon de copier des lignes d'une table vers une autre.
>
> En anglais : GB ***insert select***.

```sql
SELECT * FROM professeur;
```

```
id | nom        | prenom  | email                | actif | telephone
---+------------+---------+----------------------+-------+--------------
1  | Dubois     | Sophie  | s.dubois@ecole.be    | 1     | 0471 23 45 67
2  | Lambrechts | Ludovic | l.lambrechts@ecole.be| 1     | NULL
3  | Vermeulen  | Marc    | m.vermeulen@ecole.be | 1     | NULL
4  | Leroy      | Anne    | a.leroy@ecole.be     | 1     | NULL
5  | Martin     | Julie   | j.martin@ecole.be    | 1     | NULL
6  | Renard     | Paul    | p.renard@ecole.be    | 1     | NULL
```

Julie Martin et Paul Renard reçoivent les numéros 5 et 6, `actif` vaut `1`, `telephone` vaut `NULL`.

> ⚠️ **ET SI TU EXÉCUTES L'`INSERT … SELECT` UNE DEUXIÈME FOIS ?** SQLite refuse avec `UNIQUE constraint failed: professeur.email`. La contrainte `UNIQUE` sur l'adresse te protège contre un double import.

Supprimer la table de passage

La table `import_prof` a fait son travail. `DROP TABLE` veut dire *laisse tomber la table* : l'instruction la supprime avec son contenu.

```sql
DROP TABLE import_prof;
```

⚠️ `DROP TABLE` supprime définitivement une table et toutes ses lignes. Ici, c'est voulu : `import_prof` ne servait que de passage. Vérifie toujours le nom de la table avant d'exécuter.

Enregistrer les ajouts

Tant que tu n'as rien enregistré, tes ajouts n'existent que dans DB Browser. Le fichier `.db` sur le disque n'a pas changé.

📖 **NOUVELLE NOTION : LES MODIFICATIONS EN ATTENTE**

DB Browser garde tous tes changements **en attente** jusqu'à ce que tu cliques sur **Écrire les modifications** (`Ctrl+S`). À ce moment, ils sont enregistrés dans le fichier de la base. Tant que ce n'est pas fait, le bouton **Annuler les modifications** remet la base dans l'état du dernier enregistrement.

En anglais : GB ***Write Changes*** et GB ***Revert Changes***.

Clique sur **Écrire les modifications**. Ta base contient maintenant les 12 élèves et les 6 professeurs.

🧪 Exercices

Les exercices utilisent une table `local`. Crée-la d'abord avec ce script :

```sql
CREATE TABLE IF NOT EXISTS local (
    id         INTEGER PRIMARY KEY,
    code       TEXT    NOT NULL UNIQUE,
    etage      INTEGER,
    places     INTEGER NOT NULL,
    projecteur INTEGER NOT NULL DEFAULT 0
);
```

Exercice 1 — Prévoir le contenu de la table ★☆☆☆☆

Sans exécuter, écris sur papier ce que contiendra la table `local` après ces trois instructions : les valeurs de `id`, `etage` et `projecteur` pour chaque ligne. Exécute ensuite, puis compare avec un `SELECT`.

```sql
INSERT INTO local (code, etage, places, projecteur) VALUES ('B14', 1, 24, 1);
INSERT INTO local (code, etage, places) VALUES ('B15', 1, 20), ('A02', 0, 30);
INSERT INTO local (code, places) VALUES ('C01', 16);
```

Exercice 2 — Lire et corriger ★★☆☆☆

Chaque instruction est refusée. Exécute-la, recopie le message, explique l'erreur en une phrase, puis écris la version correcte.

```sql
-- a)
INSERT INTO local (code, etage, places) VALUES (B16, 1, 24);

-- b)
INSERT INTO local (code, etage, places) VALUES ('B16', 1);

-- c)
INSERT INTO locaux (code, etage, places) VALUES ('B16', 1, 24);

-- d)
INSERT INTO local (code, etage, places) VALUES ('B16', 1, 24),;

-- e)
INSERT INTO local VALUES ('B16', 1, 24, 0);
```

Exercice 3 — Provoquer les refus ★★☆☆☆

1. Écris un `INSERT` refusé par la contrainte `NOT NULL` de la table `local`. Recopie le message.
2. Écris un `INSERT` refusé par la contrainte `UNIQUE`. Recopie le message.
3. Exécute `INSERT INTO local (code, etage, places) VALUES ('b14', 1, 24);`. Pourquoi cette ligne est-elle acceptée alors que le local `B14` existe ? Quel problème cela pose-t-il ?

Exercice 4 — Une apostrophe dans le nom ★★★☆☆

Une nouvelle professeure s'appelle Claire D'Hondt, adresse `c.dhondt@ecole.be`.

1. Écris l'`INSERT` naturellement et exécute-le. Pourquoi SQLite est-il perdu ?
2. En SQL, pour écrire une apostrophe à l'intérieur d'un texte, on la double : `'D''Hondt'`. Corrige ton instruction, puis vérifie avec un `SELECT` que le nom est bien enregistré avec une seule apostrophe.

Exercice 5 — Importer ses propres données ★★★☆☆

1. Avec le Bloc-notes, crée un fichier `nouveaux-eleves.csv` qui contient la ligne d'en-tête `nom;prenom;commune` et deux élèves inventés.
2. Importe-le dans une table de passage `import_eleve`.
3. Copie les deux élèves dans la table `eleve` avec un `INSERT … SELECT`.
4. Vérifie le résultat : quelles valeurs ont reçu `id`, `email` et `absences` ? Pourquoi ?
5. Supprime la table de passage.

✍️ À retenir

- `INSERT INTO table (colonnes) VALUES (valeurs)` ajoute une ligne ; les textes s'écrivent entre apostrophes.
- **Nomme toujours les colonnes** : chaque valeur va dans la colonne de même position dans la liste.
- Une colonne non citée reçoit sa valeur `DEFAULT`, sinon `NULL` ; `INTEGER PRIMARY KEY` reçoit le numéro suivant.
- Plusieurs lignes d'un coup : des groupes `(…)` séparés par des virgules après `VALUES`.
- Un refus cite la contrainte, puis `table.colonne` : lis-le avant de corriger. Une ligne refusée n'est pas ajoutée.
- Import CSV : fichier → table de passage → `INSERT … SELECT` → `DROP TABLE` de la table de passage.
- Rien n'est enregistré dans le fichier tant que tu n'as pas cliqué sur **Écrire les modifications**.

Suite

La table est remplie, mais les données changent : un professeur quitte l'école, un élève a une absence de plus. L'article Modifier des lignes : UPDATE montre comment modifier une ligne sans toucher aux autres.