Modifier des lignes : UPDATE

Une commune mal écrite, une absence à ajouter, un professeur qui quitte l'école : tu modifies des lignes existantes avec UPDATE, en vérifiant d'abord par un SELECT lesquelles seront touchées.

4TTR 5TTR 6TTR Découverte

Aucune base ne reste juste bien longtemps. Une commune a été mal encodée, un élève a une absence de plus, un professeur quitte l'école. Supprimer la ligne pour la réécrire serait maladroit et risqué : on la modifie sur place, en visant exactement la bonne ligne.

> 💡 **BASE DE DÉPART**
>
> Télécharge `mon-ecole-professeurs.db` dans la barre latérale et ouvre-la avec DB Browser for SQLite. Elle contient deux tables : `eleve` (12 élèves) et `professeur` (6 professeurs). Le SQL s'écrit dans l'onglet **Exécuter le SQL**.

🎯 Objectifs

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

1. **Modifier** une ou plusieurs colonnes avec `UPDATE … SET … WHERE`.
2. **Appliquer** la méthode « `SELECT` d'abord » pour savoir quelles lignes seront touchées.
3. **Cibler** une ligne par sa clé primaire plutôt que par son nom.
4. **Calculer** une nouvelle valeur à partir de l'ancienne.
5. **Annoncer** le nombre de lignes touchées avant d'exécuter, et réagir si le nombre est faux.
6. **Annuler** une modification ratée tant qu'elle n'est pas enregistrée.

Une faute dans la table eleve

Affiche la commune de chaque élève :

```
SELECT id, nom, prenom, commune FROM eleve;
```

© 2025 Mr. Lambrechts

```
id | nom       | prenom   | commune
---+-----------+----------+----------
1  | Adam      | Lucas    | Jodoigne
2  | Bastin    | Emma     | Wavre
3  | Charlier  | Noah     | Perwez
4  | Delvaux   | Léa      | Jodoigne
5  | Englebert | Hugo     | Hannut
6  | Fontaine  | Chloé    | Jodoigne
7  | Gérard    | Nathan   | Wavre
8  | Hubert    | Manon    | Perwez
9  | Istas     | Louis    | Jodoigne
10 | Jacques   | Camille  | Hannut
11 | Kevers    | Arthur   | Ramillies
12 | Lambert   | Sarah    | jodoigne
```

Sarah Lambert habite `jodoigne`, en minuscules. Pour SQLite, `jodoigne` et `Jodoigne` sont deux textes différents : une recherche des élèves de `'Jodoigne'` oublierait Sarah. Il faut corriger cette case.

La méthode : le SELECT d'abord

Une modification touche **toutes** les lignes qui répondent à la condition. Avant de modifier, on vérifie donc quelles lignes répondent à cette condition, avec un `SELECT`.

Étape 1 : trouver la ligne

```sql
SELECT id, nom, prenom, commune
FROM eleve
WHERE id = 12;
```

```
id | nom     | prenom | commune
---+---------+--------+---------
12 | Lambert | Sarah  | jodoigne
```

Une ligne, et c'est la bonne. `WHERE` veut dire *où* : on garde les lignes où la condition est vraie.

Étape 2 : transformer le SELECT en UPDATE

On garde le `WHERE` tel quel, et on remplace le début :

```sql
UPDATE eleve
SET commune = 'Jodoigne'
WHERE id = 12;
```

`UPDATE` veut dire *mets à jour*, et `SET` *règle* ou *fixe* : « mets à jour la table `eleve`, règle `commune` sur `'Jodoigne'`, pour les lignes où `id` vaut 12 ».

© 2025 Mr. Lambrechts

> 📖 **NOUVELLE NOTION : UPDATE**
>
> L'instruction `UPDATE table SET colonne = valeur WHERE condition` modifie des lignes qui existent déjà. Elle remplace la valeur de la colonne dans toutes les lignes où la condition est vraie. Les autres lignes et les autres colonnes ne changent pas.
>
> En anglais : GB ***update***, *mettre à jour*.

Étape 3 : lire le nombre de lignes touchées

Après l'exécution, DB Browser indique sous la zone de requête le nombre de lignes affectées par l'instruction. Ici : une.

> 📖 **NOUVELLE NOTION : LE NOMBRE DE LIGNES AFFECTÉES**
>
> Le **nombre de lignes affectées** est le nombre de lignes qu'une instruction `INSERT`, `UPDATE` ou `DELETE` a réellement touchées. DB Browser l'affiche après chaque exécution. C'est le premier indice qu'une modification a fait ce qu'on voulait, ou pas.
>
> En anglais : GB ***rows affected***.

Étape 4 : vérifier

```sql
SELECT id, nom, prenom, commune
FROM eleve
WHERE id = 12;
```

```
id | nom     | prenom | commune
---+---------+--------+---------
12 | Lambert | Sarah  | Jodoigne
```

Et la recherche des élèves de Jodoigne retrouve enfin Sarah :

```sql
SELECT id, nom, prenom, commune
FROM eleve
WHERE commune = 'Jodoigne';
```

```
id | nom      | prenom | commune
---+----------+--------+---------
1  | Adam     | Lucas  | Jodoigne
4  | Delvaux  | Léa    | Jodoigne
6  | Fontaine | Chloé  | Jodoigne
9  | Istas    | Louis  | Jodoigne
12 | Lambert  | Sarah  | Jodoigne
```

Pourquoi viser l'id plutôt que le nom

© 2025 Mr. Lambrechts

On aurait pu écrire `WHERE nom = 'Lambert'`. Aujourd'hui, cela touche la bonne ligne. Mais le jour où une autre élève Lambert s'inscrit, la même instruction modifie les deux. Le nom décrit une personne, il ne la désigne pas à coup sûr.

> 🧠 **POUR MODIFIER UNE LIGNE PRÉCISE, VISE SA CLÉ PRIMAIRE.** L'`id` est unique et ne change jamais : `WHERE id = 12` touche une ligne et une seule, aujourd'hui comme dans dix ans. Le `SELECT` de l'étape 1 sert justement à trouver cet `id`.

Calculer à partir de l'ancienne valeur

Louis Istas a été absent ce matin. Son compteur doit augmenter de un.

```sql
SELECT id, nom, prenom, absences
FROM eleve
WHERE id = 9;
```

```
id | nom   | prenom | absences
---+-------+--------+---------
9  | Istas | Louis  | 12
```

Tu pourrais écrire `SET absences = 13`. Mais il faudrait d'abord lire la valeur, et le jour où quelqu'un l'a changée entre-temps, ton `13` serait faux. On écrit plutôt le calcul :

```sql
UPDATE eleve
SET absences = absences + 1
WHERE id = 9;
```

À droite du `=`, `absences` désigne l'**ancienne** valeur de la ligne. SQLite la lit, ajoute 1, puis range le résultat dans la colonne.

```sql
SELECT id, nom, prenom, absences
FROM eleve
WHERE id = 9;
```

```
id | nom   | prenom | absences
---+-------+--------+---------
9  | Istas | Louis  | 13
```

Modifier les professeurs

© 2025 Mr. Lambrechts

Un numéro de téléphone

Ludovic Lambrechts communique son numéro. On vérifie la ligne :

```sql
SELECT id, nom, prenom, telephone
FROM professeur
WHERE id = 2;
```

```
id | nom        | prenom  | telephone
---+------------+---------+----------
2  | Lambrechts | Ludovic | NULL
```

Puis on modifie. Le téléphone est du texte : il s'écrit entre apostrophes.

```sql
UPDATE professeur
SET telephone = '0485 11 22 33'
WHERE id = 2;
```

Plusieurs colonnes dans un même SET

Paul Renard quitte l'école. On le marque comme inactif (`actif = 0`) et on s'assure qu'aucun numéro de téléphone ne reste enregistré à son nom. Deux colonnes changent : on les sépare par une **virgule** dans le même `SET`.

```sql
SELECT id, nom, prenom, actif, telephone
FROM professeur
WHERE id = 6;
```

```
id | nom    | prenom | actif | telephone
---+--------+--------+-------+----------
6  | Renard | Paul   | 1     | NULL
```

```sql
UPDATE professeur
SET actif = 0,
    telephone = NULL
WHERE id = 6;
```

```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     | 0485 11 22 33
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      | 0     | NULL
```

© 2025 Mr. Lambrechts

> ⚠️ **UNE VIRGULE, PAS `AND`.** `SET commune = 'Hannut' AND absences = 1` ne provoque aucune erreur. Mais SQLite le comprend comme un seul calcul logique, dont le résultat (`0`) est rangé dans `commune` : la commune devient `0` et les absences ne changent pas. Dans un `SET`, les colonnes se séparent par des virgules.

Toutes ces corrections sont bonnes. Clique sur **Écrire les modifications** (`Ctrl+S`) pour les enregistrer dans le fichier, avant l'expérience qui suit.

L'expérience du WHERE oublié

DB Browser garde tes changements en attente jusqu'à ce que tu cliques sur **Écrire les modifications**. Tant que ce n'est pas fait, **Annuler les modifications** remet la base dans l'état du dernier enregistrement. C'est ce filet de sécurité qui permet l'expérience suivante.

> 📖 **NOUVELLE NOTION : ÉCRIRE OU ANNULER LES MODIFICATIONS**
>
> Dans DB Browser, **Écrire les modifications** enregistre dans le fichier tous les changements en attente. **Annuler les modifications** les abandonne et revient au dernier enregistrement. Une fois les modifications écrites, il n'est plus possible de revenir en arrière.
>
> En anglais : GB ***Write Changes*** et GB ***Revert Changes***.

Imagine que tu veuilles corriger la commune de Sarah, et que tu oublies le `WHERE`. Exécute :

```
UPDATE eleve
SET commune = 'Jodoigne';
```

Aucune erreur. Regarde le nombre de lignes affectées annoncé par DB Browser : **douze**. Toutes les lignes de la table.

```
SELECT id, nom, prenom, commune FROM eleve;
```

```
id | nom       | prenom  | commune
---+-----------+---------+---------
1  | Adam      | Lucas   | Jodoigne
2  | Bastin    | Emma    | Jodoigne
3  | Charlier  | Noah    | Jodoigne
4  | Delvaux   | Léa     | Jodoigne
5  | Englebert | Hugo    | Jodoigne
6  | Fontaine  | Chloé   | Jodoigne
7  | Gérard    | Nathan  | Jodoigne
8  | Hubert    | Manon   | Jodoigne
9  | Istas     | Louis   | Jodoigne
10 | Jacques   | Camille | Jodoigne
```

© 2025 Mr. Lambrechts

```
11 | Kevers    | Arthur  | Jodoigne
12 | Lambert   | Sarah   | Jodoigne
```

Emma habite soudain Jodoigne, Arthur aussi, et plus personne n'habite Wavre, Perwez, Hannut ni Ramillies. Sept élèves ont « déménagé » en une seconde. Si tu écrivais ces modifications maintenant, il faudrait retrouver l'ancienne commune de chacun, un par un.

Clique sur **Annuler les modifications**, puis relance le `SELECT` : chaque élève a retrouvé sa commune.

> 🧠 **UN `UPDATE` SANS `WHERE` MODIFIE TOUTES LES LIGNES DE LA TABLE.** SQLite ne demande aucune confirmation : pour lui, c'est une instruction parfaitement valable.

> 💡 **ANNONCE LE NOMBRE AVANT D'EXÉCUTER.** Avant chaque `UPDATE`, dis à voix haute combien de lignes il doit toucher (grâce au `SELECT` d'abord). Après l'exécution, compare avec le nombre annoncé par DB Browser. S'ils ne correspondent pas, n'écris pas les modifications : annule, et cherche pourquoi.

Quand la base refuse la modification

Les contraintes de la table jouent aussi pendant un `UPDATE`.

Marc Vermeulen (`id` 3) reçoit par erreur l'adresse de Sophie Dubois :

```sql
UPDATE professeur
SET email = 's.dubois@ecole.be'
WHERE id = 3;
```

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

Et on essaie d'effacer son prénom :

```sql
UPDATE professeur
SET prenom = NULL
WHERE id = 3;
```

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

Le message cite la contrainte, puis la table et la colonne en cause. La ligne, elle, n'a pas bougé :

```sql
SELECT * FROM professeur WHERE id = 3;
```

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

© 2025 Mr. Lambrechts

```
3 | 3  | Vermeulen | Marc   | m.vermeulen@ecole.be | 1     | NULL
```

> ⚠️ **ZÉRO LIGNE N'EST PAS UNE ERREUR.** `UPDATE professeur SET telephone = '0477 00 00 00' WHERE id = 42;` s'exécute sans message d'erreur, mais aucun professeur ne porte le numéro 42 : DB Browser annonce zéro ligne affectée. Si tu attendais une ligne, c'est ta condition qui est fausse.

La base est maintenant corrigée. Si tu as fait des essais depuis ton dernier enregistrement, clique sur **Annuler les modifications**.

🧪 Exercices

Pars de ta base à la fin de l'article. Après chaque essai qui modifie des données, clique sur **Annuler les modifications**.

Exercice 1 — Annoncer le nombre de lignes ★☆☆☆☆

Pour chaque instruction, écris **avant de l'exécuter** combien de lignes elle va toucher. Écris le `SELECT` qui te permet de le savoir, exécute l'`UPDATE`, compare avec DB Browser, puis annule.

```sql
-- a)
UPDATE eleve SET absences = absences + 1 WHERE commune = 'Wavre';

-- b)
UPDATE eleve SET absences = absences + 1 WHERE absences >= 4;

-- c)
UPDATE professeur SET telephone = '0400 00 00 00' WHERE actif = 1;

-- d)
UPDATE eleve SET absences = 0 WHERE id = 42;

-- e)
UPDATE eleve SET absences = 0;
```

Question bonus : `UPDATE eleve SET commune = 'Jodoigne' WHERE commune = 'Jodoigne';` ne change aucune valeur. Combien de lignes DB Browser annonce-t-il ? Qu'en conclus-tu sur ce que compte ce nombre ?

Exercice 2 — De la demande à l'UPDATE ★★☆☆☆

Pour chaque demande : écris le `SELECT` qui trouve la ligne, note son `id`, écris l'`UPDATE`, puis vérifie.

1. Hugo Englebert a déménagé à Wavre.
2. Noah Charlier a enfin une adresse : `noah.charlier@ecole.be`.
3. Anne Leroy donne son numéro : `0472 98 76 54`.
4. Camille Jacques a été absente deux jours de plus.

Exercice 3 — Lire et corriger ★★☆☆☆

Exécute chaque instruction, recopie le message, trouve l'erreur et écris la version correcte.

```sql
-- a)
UPDATE eleve SET commune = Wavre WHERE id = 2;

-- b)
UPDATE eleve commune = 'Wavre' WHERE id = 2;

-- c)
UPDATE eleve SET absences = absences + 1, WHERE id = 2;
```

La dernière ne produit **aucun** message. Exécute-la, affiche la ligne 2 avec un `SELECT`, décris ce qui s'est passé, puis annule et corrige.

```sql
-- d)
UPDATE eleve SET commune = 'Hannut' AND absences = 1 WHERE id = 2;
```

Exercice 4 — Plusieurs lignes à la fois ★★★☆☆

Une grève des bus a empêché les élèves de Perwez de venir ce matin. Ajoute une absence à chacun d'eux.

1. Annonce le nombre de lignes qui seront touchées, et écris le `SELECT` qui le prouve.
2. Écris l'`UPDATE` et vérifie que le nombre annoncé par DB Browser correspond.
3. Pourquoi ne vise-t-on pas l'`id` cette fois ?

Exercice 5 — Désactiver plutôt que supprimer ★★★☆☆

Pour Paul Renard, on a mis `actif = 0` au lieu de supprimer sa ligne.

1. Donne deux raisons pour lesquelles l'école peut vouloir garder la trace d'un professeur qui est parti.
2. Écris la requête qui affiche uniquement les professeurs encore en service.
3. Marc Vermeulen part à la retraite. Écris l'instruction complète, avec sa vérification.

✍️ À retenir

- `UPDATE table SET colonne = valeur WHERE condition` modifie toutes les lignes où la condition est vraie.
- **Le `SELECT` d'abord** : écris la condition dans un `SELECT`, compte les lignes, puis transforme-le en `UPDATE` en gardant le même `WHERE`.
- Pour une ligne précise, vise sa clé primaire : `WHERE id = …`.
- `SET absences = absences + 1` calcule à partir de l'ancienne valeur ; plusieurs colonnes se séparent par des **virgules**, jamais par `AND`.
- **Sans `WHERE`, toutes les lignes sont modifiées**, sans confirmation.

© 2025 Mr. Lambrechts

- Annonce le nombre de lignes avant d'exécuter ; s'il ne correspond pas, **Annuler les modifications** avant d'écrire.
- Une contrainte violée fait refuser l'`UPDATE` ; zéro ligne affectée n'est pas une erreur, mais un signal.

Suite

Parfois, une ligne ne doit pas être corrigée mais effacée. L'article Supprimer des lignes : DELETE commence par une sauvegarde, parce qu'une suppression enregistrée ne se rattrape pas.

© 2025 Mr. Lambrechts