Filtrer les lignes avec WHERE

Les élèves de Wavre, ceux qui ont plus de quatre absences, ceux nés avant 2010 : WHERE ne garde que les lignes qui remplissent une condition.

4TTR 5TTR 6TTR Découverte

Quels élèves ont manqué plus de quatre demi-jours ? Qui habite à Perwez ou à Hannut ? Pour répondre, tu n'as pas besoin de toutes les lignes de la table, seulement de certaines. Il faut donc dire à la base lesquelles garder.

> 💡 Télécharge `mon-ecole-eleves.db` dans la sidebar, ouvre-la dans DB Browser for SQLite, puis va dans l'onglet **Exécuter le SQL**. Tape chaque requête dans la zone du haut et exécute-la avec ▶ ou Ctrl+Entrée.

🎯 Objectifs

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

1. filtrer les lignes d'une table avec `WHERE` ;
2. utiliser les opérateurs `=`, `<>`, `<`, `>`, `<=` et `>=` ;
3. savoir quand mettre des apostrophes autour d'une valeur ;
4. combiner des conditions avec `AND` et `OR`, en plaçant les parenthèses nécessaires ;
5. utiliser `BETWEEN`, `IN` et `NOT IN` ;
6. expliquer pourquoi une faute de frappe peut faire disparaître une ligne d'un résultat.

Garder seulement certaines lignes

La base contient une table `eleve`, avec une ligne par élève. Pour la lire, on écrit une **requête** : `SELECT` (*sélectionne*) donne les colonnes à afficher, `FROM` (*depuis*) donne la table où les chercher.

Une requête `SELECT … FROM …` renvoie toutes les lignes. Pour ne garder qu'Emma, on ajoute une troisième partie :

```sql
SELECT nom, prenom
FROM eleve
WHERE prenom = 'Emma';
```

```
nom    | prenom
-------+-------
Bastin | Emma
```

`WHERE` veut dire *où*. La requête se lit presque comme une phrase : « sélectionne le nom et le prénom depuis la table `eleve`, **où** le prénom est égal à Emma ».

SQLite examine les lignes une par une. Pour chacune, il vérifie si `prenom = 'Emma'` est vrai. Si oui, la ligne est gardée. Sinon, elle est écartée.

> 📖 **NOUVELLE NOTION : LA CONDITION**
>
> Une **condition** est une affirmation que SQLite peut vérifier sur chaque ligne, et qui est soit vraie, soit fausse. `WHERE` (*où*) ne garde que les lignes pour lesquelles la condition est vraie. Les autres lignes n'apparaissent pas dans le résultat, mais restent dans la table.
>
> En anglais : GB ***condition***.

Les opérateurs de comparaison

Le signe `=` n'est pas le seul moyen de comparer. Pour trouver les élèves qui ont plus de quatre demi-jours d'absence, on utilise `>` :

```sql
SELECT prenom, nom, absences
FROM eleve
WHERE absences > 4;
```

```
prenom | nom       | absences
-------+-----------+---------
Noah   | Charlier  | 5
Hugo   | Englebert | 8
Louis  | Istas     | 12
```

Camille Jacques, qui a exactement 4 absences, n'est pas dans la liste : `>` veut dire *strictement plus grand*. Pour l'inclure, il faudrait écrire `>=`.

> 📖 **NOUVELLE NOTION : L'OPÉRATEUR DE COMPARAISON**
>
> Un **opérateur de comparaison** est un symbole qui compare deux valeurs : `=`, `<>`, `<`, `>`, `<=`, `>=`. Le résultat de la comparaison est vrai ou faux. C'est la brique de base de toute condition.
>
> En anglais : GB ***comparison operator***.

Opérateur	Se lit	Exemple
`=`	égal à	`WHERE commune = 'Wavre'`
`<>`	différent de	`WHERE commune <> 'Wavre'`
`<`	strictement plus petit que	`WHERE absences < 3`
`>`	strictement plus grand que	`WHERE absences > 4`
`<=`	plus petit ou égal à	`WHERE absences <= 1`
`>=`	plus grand ou égal à	`WHERE date_naissance >= '2010-06-01'`

Les opérateurs `<` et `>` fonctionnent aussi sur les dates. Voici les élèves nés avant 2010 :

```sql
SELECT prenom, nom, date_naissance
FROM eleve
WHERE date_naissance < '2010-01-01';
```

```
prenom | nom      | date_naissance
-------+----------+---------------
Noah   | Charlier | 2009-11-23
Chloé  | Fontaine | 2009-09-08
Louis  | Istas    | 2009-12-11
Arthur | Kevers   | 2009-10-17
```

Les dates sont stockées comme du texte, au format `AAAA-MM-JJ`. Comme l'année est écrite en premier, comparer deux de ces textes revient à comparer les dates.

> 💡 `<>` est la façon standard d'écrire « différent de » en SQL. SQLite accepte aussi `!=`, que tu rencontreras dans d'autres langages.

Des apostrophes autour du texte et des dates

Dans les exemples précédents, `'Emma'` et `'2010-01-01'` sont entourés d'apostrophes, mais pas `4`. Ce n'est pas un détail.

Type de valeur	Écriture	Exemple
Texte	entre apostrophes	`prenom = 'Emma'`
Date (stockée en texte)	entre apostrophes	`date_naissance < '2010-01-01'`

Type de valeur	Écriture	Exemple
Nombre	sans apostrophes	`absences > 4`

Les apostrophes disent à SQLite : « ceci est une valeur, pas un nom de colonne ». Regarde ce qui se passe quand on les oublie autour d'un prénom :

```
SELECT nom
FROM eleve
WHERE prenom = Emma;
```

```
no such column: Emma
```

Sans apostrophes, SQLite croit que `Emma` est le nom d'une colonne. Il la cherche, ne la trouve pas, et répond *pas de colonne de ce nom*. L'erreur est au moins visible.

Oublier les apostrophes autour d'une date est bien plus sournois :

```
SELECT prenom, nom, date_naissance
FROM eleve
WHERE date_naissance > 2010-01-01;
```

```
prenom  | nom       | date_naissance
--------+-----------+---------------
Lucas   | Adam      | 2010-03-12
Emma    | Bastin    | 2010-07-04
Noah    | Charlier  | 2009-11-23
Léa     | Delvaux   | 2010-04-15
Hugo    | Englebert | 2010-01-30
Chloé   | Fontaine  | 2009-09-08
Nathan  | Gérard    | 2010-05-19
Manon   | Hubert    | 2010-02-27
Louis   | Istas     | 2009-12-11
Camille | Jacques   | 2010-06-02
Arthur  | Kevers    | 2009-10-17
Sarah   | Lambert   | 2010-09-01
```

Aucune erreur, et pourtant les élèves nés en 2009 sont là. Sans apostrophes, `2010-01-01` n'est plus une date : c'est un calcul, 2010 moins 1 moins 1.

```
SELECT 2010-01-01 AS calcul;
```

```
calcul
------
2008
```

SQLite calcule donc 2008. Comme la colonne `date_naissance` contient du texte, il transforme ce nombre en texte `'2008'`, puis compare caractère par caractère. Toutes les dates commencent par `2009` ou `2010`, qui viennent après `2008`. La condition est vraie pour toutes les lignes.

⚠️ Une requête qui provoque une erreur se corrige facilement. Une requête qui renvoie un résultat faux sans rien dire est bien plus dangereuse : vérifie toujours que le résultat a du sens.

ℹ️ SQLite est tolérant : `absences > '4'` , avec des apostrophes, fonctionne aussi, parce qu'il convertit la valeur. D'autres logiciels de base de données sont plus stricts. Prends l'habitude de ne pas mettre d'apostrophes autour des nombres.

Combiner plusieurs conditions avec AND et OR

Une seule condition ne suffit pas toujours. Pour trouver les élèves de Wavre qui ont au moins une absence, il faut deux conditions à la fois. On les relie avec `AND`, qui veut dire *et* :

```sql
SELECT prenom, nom, absences
FROM eleve
WHERE commune = 'Wavre'
  AND absences > 0;
```

```
prenom | nom    | absences
-------+--------+---------
Nathan | Gérard | 3
```

Emma Bastin habite aussi Wavre, mais elle n'a aucune absence : la deuxième condition est fausse pour elle, donc la ligne est écartée.

Pour les élèves qui habitent Perwez **ou** Hannut, une seule des deux conditions doit être vraie. On utilise `OR`, qui veut dire *ou* :

```sql
SELECT prenom, nom, commune
FROM eleve
WHERE commune = 'Perwez'
   OR commune = 'Hannut';
```

```
prenom  | nom       | commune
--------+-----------+--------
Noah    | Charlier  | Perwez
Hugo    | Englebert | Hannut
Manon   | Hubert    | Perwez
Camille | Jacques   | Hannut
```

Opérateur	Traduction	La ligne est gardée si…
`AND`	et	**toutes** les conditions sont vraies
`OR`	ou	**au moins une** condition est vraie

⚠️ Chaque côté de `AND` ou de `OR` doit être une condition complète. `WHERE nom = 'Adam' OR 'Bastin'` ne veut pas dire « Adam ou Bastin » : il faut écrire `WHERE nom = 'Adam' OR nom = 'Bastin'` .

Quand AND et OR se mélangent

Voici une question qui mélange les deux : « les élèves de Perwez ou de Hannut qui ont plus de 3 absences ». Une première traduction vient naturellement :

```sql
SELECT prenom, nom, commune, absences
FROM eleve
WHERE commune = 'Perwez'
   OR commune = 'Hannut'
  AND absences > 3;
```

```
prenom  | nom       | commune | absences
--------+-----------+---------+---------
Noah    | Charlier  | Perwez  | 5
Hugo    | Englebert | Hannut  | 8
Manon   | Hubert    | Perwez  | 0
Camille | Jacques   | Hannut  | 4
```

Manon Hubert n'a aucune absence. Elle ne devrait pas être là. Aucune erreur n'a été signalée : la requête est valide, mais elle ne pose pas la question voulue.

La raison : SQLite évalue **toujours** `AND` **avant** `OR`, comme en mathématiques on calcule la multiplication avant l'addition. La condition a donc été comprise comme ceci :

```sql
commune = 'Perwez'  OR  (commune = 'Hannut' AND absences > 3)
```

Autrement dit : « tous les élèves de Perwez, plus ceux de Hannut qui ont plus de 3 absences ». Manon habite Perwez, elle est donc gardée.

📖 NOUVELLE NOTION : LA PRIORITÉ DE AND SUR OR

La **priorité** fixe l'ordre dans lequel SQL évalue les opérateurs d'une condition. `AND` est prioritaire sur `OR` : il est évalué en premier. Pour imposer un autre ordre, on entoure de parenthèses la partie à évaluer d'abord.

> En anglais : GB ***operator precedence***.

Avec des parenthèses, on regroupe les deux communes avant d'appliquer la condition sur les absences :

```sql
SELECT prenom, nom, commune, absences
FROM eleve
WHERE (commune = 'Perwez' OR commune = 'Hannut')
  AND absences > 3;
```

```
prenom  | nom       | commune | absences
--------+-----------+---------+---------
Noah    | Charlier  | Perwez  | 5
Hugo    | Englebert | Hannut  | 8
Camille | Jacques   | Hannut  | 4
```

> 🧠 Dès qu'une condition mélange `AND` et `OR`, mets des parenthèses. Même quand elles ne changent rien, elles rendent ton intention lisible.

Chercher dans un intervalle avec BETWEEN

Pour les élèves qui ont entre 1 et 4 absences, tu pourrais écrire `absences >= 1 AND absences <= 4`. SQL propose une écriture plus courte : `BETWEEN … AND …`, qui veut dire *entre … et …*.

```sql
SELECT prenom, nom, absences
FROM eleve
WHERE absences BETWEEN 1 AND 4;
```

```
prenom  | nom     | absences
--------+---------+---------
Lucas   | Adam    | 2
Léa     | Delvaux | 1
Nathan  | Gérard  | 3
Camille | Jacques | 4
Arthur  | Kevers  | 1
```

Léa Delvaux (1 absence) et Camille Jacques (4 absences) sont dans le résultat : avec `BETWEEN`, **les deux bornes sont comprises**.

`BETWEEN` fonctionne aussi avec les dates. Voici les élèves nés pendant le premier trimestre 2010 :

```sql
SELECT prenom, nom, date_naissance
FROM eleve
WHERE date_naissance BETWEEN '2010-01-01' AND '2010-03-31';
```

```
prenom | nom       | date_naissance
-------+-----------+---------------
Lucas  | Adam      | 2010-03-12
Hugo   | Englebert | 2010-01-30
Manon  | Hubert    | 2010-02-27
```

⚠️ La plus petite valeur s'écrit en premier. `BETWEEN 4 AND 1` ne renvoie aucune ligne, sans message d'erreur : aucun nombre n'est à la fois plus grand que 4 et plus petit que 1.

Comparer à une liste avec IN et NOT IN

Pour les élèves de Perwez, de Hannut ou de Ramillies, trois `OR` à la suite deviennent vite pénibles. `IN`, qui veut dire *dans*, compare une colonne à une liste de valeurs :

```sql
SELECT prenom, nom, commune
FROM eleve
WHERE commune IN ('Perwez', 'Hannut', 'Ramillies');
```

```
prenom  | nom       | commune
--------+-----------+----------
Noah    | Charlier  | Perwez
Hugo    | Englebert | Hannut
Manon   | Hubert    | Perwez
Camille | Jacques   | Hannut
Arthur  | Kevers    | Ramillies
```

La ligne est gardée si la commune fait partie de la liste. À l'inverse, `NOT IN` (*pas dans*) garde les lignes dont la valeur n'est pas dans la liste. Cherche les élèves qui n'habitent ni Jodoigne ni Wavre :

```sql
SELECT prenom, nom, commune
FROM eleve
WHERE commune NOT IN ('Jodoigne', 'Wavre');
```

```
prenom  | nom       | commune
--------+-----------+----------
Noah    | Charlier  | Perwez
Hugo    | Englebert | Hannut
Manon   | Hubert    | Perwez
Camille | Jacques   | Hannut
Arthur  | Kevers    | Ramillies
Sarah   | Lambert   | jodoigne
```

Regarde la dernière ligne. Sarah Lambert habite Jodoigne, et pourtant elle est dans la liste de ceux qui n'y habitent pas.

Une faute de frappe qui échappe au signe égal

Compte les élèves de Jodoigne :

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

```
prenom | nom
-------+---------
Lucas  | Adam
Léa    | Delvaux
Chloé  | Fontaine
Louis  | Istas
```

Quatre élèves. Pourtant, dans l'onglet **Parcourir les données**, cinq élèves habitent Jodoigne. La cinquième, Sarah Lambert, a été encodée avec une faute : `jodoigne`, en minuscules.

Le signe `=` compare deux textes caractère par caractère. `J` majuscule et `j` minuscule sont deux caractères différents, donc `'jodoigne' = 'Jodoigne'` est faux. La ligne est écartée.

Tu peux contourner le problème en acceptant les deux écritures :

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

```
prenom | nom      | commune
-------+----------+---------
Lucas  | Adam     | Jodoigne
Léa    | Delvaux  | Jodoigne
Chloé  | Fontaine | Jodoigne
Louis  | Istas    | Jodoigne
Sarah  | Lambert  | jodoigne
```

🧠 Une condition ne retrouve que ce qui est écrit exactement comme elle le demande. Une faute de frappe dans les données ne provoque aucune erreur : la ligne disparaît simplement du résultat.

ℹ️ Contourner la faute dans chaque requête n'est pas une solution durable. La vraie solution est de corriger la donnée elle-même, avec une requête de modification (`UPDATE`). En attendant, `LIKE` offre une recherche plus souple : c'est le sujet de l'article suivant.

🧪 Exercices

Pour chaque exercice, écris la requête, exécute-la, puis vérifie que le résultat répond vraiment à la question.

Exercice 1 — Comparer ★☆☆☆☆

Écris une requête qui affiche le prénom et le nom :

1. d'Arthur ;
2. des élèves qui n'ont aucune absence ;
3. des élèves nés en 2009 ;
4. des élèves qui n'habitent pas Wavre ;
5. des élèves qui ont au moins 5 absences.

Exercice 2 — Combiner des conditions ★★☆☆☆

Écris une requête qui affiche le prénom, le nom et les colonnes utiles pour vérifier le résultat :

1. les élèves de Jodoigne qui ont plus de 2 absences ;
2. les élèves nés en 2010 qui habitent Hannut ou Wavre ;
3. les élèves qui ont entre 3 et 8 absences, d'abord avec `BETWEEN`, puis avec `AND` ;
4. les élèves de Perwez, Wavre ou Ramillies, avec `IN`.

Exercice 3 — Prédire le résultat ★★☆☆☆

Sans exécuter, écris combien de lignes renvoie chaque requête, et lesquelles. Exécute ensuite pour vérifier.

```sql
-- a)
SELECT nom, prenom FROM eleve WHERE absences > 2 AND absences < 5;

-- b)
SELECT nom, prenom FROM eleve WHERE commune = 'Wavre' OR absences = 0 AND commune = 'Perwez';

-- c)
SELECT nom, prenom FROM eleve WHERE date_naissance BETWEEN '2010-06-30' AND '2010-01-01';

-- d)
SELECT nom FROM eleve WHERE commune NOT IN ('Jodoigne');
```

Exercice 4 — Réparer des requêtes ★★★☆☆

Chacune de ces requêtes provoque une erreur ou ne répond pas à la question. Pour chacune : exécute-la, observe, explique ce qui ne va pas, puis corrige-la.

```sql
-- a) La ligne d'Emma
SELECT nom FROM eleve WHERE prenom = Emma;
```

```sql

    -- b) Les élèves qui s'appellent Adam ou Bastin
    SELECT nom, prenom FROM eleve WHERE nom = 'Adam' OR 'Bastin';

    -- c) Les élèves nés après le 1er janvier 2010
    SELECT nom FROM eleve WHERE date_naissance > 2010-01-01;

    -- d) Les élèves de Wavre
SELECT nom, prenom FROM eleve WHERE commune = 'Wavre;

-- e) Les élèves de Perwez ou de Hannut qui n'ont aucune absence
    SELECT nom, prenom, commune, absences FROM eleve
    WHERE commune = 'Perwez' OR commune = 'Hannut' AND absences = 0;

    -- f) Les élèves qui ont entre 1 et 4 absences
    SELECT nom FROM eleve WHERE absences BETWEEN 4 AND 1;
```

Exercice 5 — L'élève introuvable ★★★☆☆

La direction veut écrire aux parents des élèves qui habitent Jodoigne.

1. Écris la requête que la direction utiliserait naturellement. Combien de lignes obtiens-tu ?
2. Ouvre l'onglet **Parcourir les données** et compte à l'œil. Combien d'élèves habitent réellement Jodoigne ?
3. Quelle famille ne recevrait pas le courrier ? Explique pourquoi en deux phrases.
4. Écris une requête qui retrouve les cinq élèves.

✍️ À retenir

- `WHERE` (*où*) ne garde que les lignes pour lesquelles la condition est vraie.
- Opérateurs de comparaison : `=`, `<>`, `<`, `>`, `<=`, `>=`.
- Le texte et les dates s'écrivent entre apostrophes, les nombres sans. Sans apostrophes, `2010-01-01` devient le calcul 2008.
- `AND` : toutes les conditions doivent être vraies. `OR` : une seule suffit.
- `AND` est évalué avant `OR` : dès qu'ils se mélangent, mets des parenthèses.
- `BETWEEN a AND b` inclut les deux bornes ; `IN (…)` et `NOT IN (…)` comparent à une liste.
- Le signe `=` compare exactement : une faute de majuscule suffit à faire disparaître une ligne.

Suite

Le signe `=` exige la valeur exacte. Pour retrouver un nom dont tu ne connais que le début, ou pour ignorer les différences de majuscules, passe à Chercher un motif avec LIKE.