Trier les résultats avec ORDER BY

Du plus âgé au plus jeune, du plus absent au moins absent, par ordre alphabétique : ORDER BY range les lignes d'un résultat dans l'ordre que tu choisis.

4TTR 5TTR 6TTR Découverte

Qui est l'élève le plus âgé ? Qui a manqué le plus de cours ? Avec douze élèves, tu peux encore chercher à l'œil. Avec douze cents, c'est impossible : il faut demander à la base de ranger les lignes à ta place.

> 💡 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. trier le résultat d'une requête avec `ORDER BY` ;
2. choisir un ordre croissant (`ASC`) ou décroissant (`DESC`) ;
3. trier sur plusieurs colonnes ;
4. prévoir comment SQLite range des nombres, des dates et du texte ;
5. placer `ORDER BY` au bon endroit dans une requête.

La table de départ

La base contient une seule table, `eleve`. Chaque ligne décrit un élève : son nom, son prénom, sa date de naissance, sa commune, son adresse email et son nombre de demi-jours d'absence.

Pour lire une table, on écrit une **requête** : une question posée à la base, en langage SQL. La plus simple a deux parties. `SELECT` (*sélectionne*) donne les colonnes à afficher, `FROM` (*depuis*) donne la table où les chercher.

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

```
nom       | prenom
----------+--------
```

© 2025 Mr. Lambrechts

```
Adam      | Lucas
Bastin    | Emma
Charlier  | Noah
Delvaux   | Léa
Englebert | Hugo
Fontaine  | Chloé
Gérard    | Nathan
Hubert    | Manon
Istas     | Louis
Jacques   | Camille
Kevers    | Arthur
Lambert   | Sarah
```

Les noms semblent déjà rangés par ordre alphabétique. C'est un hasard : les élèves ont été encodés dans cet ordre-là. Rien, dans la requête, ne demande de les trier.

Trier sur une colonne

Pour trouver l'élève le plus âgé, il faut ranger les lignes selon la date de naissance. On ajoute pour cela `ORDER BY` à la fin de la requête, suivi du nom de la colonne.

```sql
SELECT prenom, nom, date_naissance
FROM eleve
ORDER BY date_naissance;
```

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

Chloé Fontaine, née en septembre 2009, arrive en tête : c'est la plus âgée. Sarah Lambert, la plus jeune, ferme la liste.

> 📖 **NOUVELLE NOTION : TRIER AVEC ORDER BY**
>
> **Trier**, c'est ranger les lignes d'un résultat selon les valeurs d'une ou plusieurs colonnes. En SQL, on trie avec `ORDER BY`, qui se traduit par *ordonner par*. Le tri change seulement l'ordre d'affichage : le contenu de la table ne bouge pas.

© 2025 Mr. Lambrechts

En anglais : GB ***sort***.

Croissant ou décroissant

Par défaut, le tri est **croissant** : du plus petit au plus grand, de A à Z, de la date la plus ancienne à la plus récente. Pour inverser l'ordre, on ajoute `DESC` après la colonne.

```sql
SELECT prenom, nom, date_naissance
FROM eleve
ORDER BY date_naissance DESC;
```

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

`DESC` est l'abréviation de *descending*, **décroissant**. Son contraire, `ASC`, vient de *ascending*, **croissant**.

Mot-clé	Mot anglais	Traduction	Exemple d'ordre
`ASC`	GB ***ascending***	croissant	0, 1, 2… / A, B, C…
`DESC`	GB ***descending***	décroissant	12, 8, 5… / Z, Y, X…

🧠 Sans précision, le tri est croissant. `ORDER BY date_naissance` et `ORDER BY date_naissance ASC` donnent exactement le même résultat.

Ce que SQLite compare selon le type de données

Trier, c'est comparer les valeurs deux à deux. Or SQLite ne compare pas un nombre, une date et un texte de la même façon. Il vaut la peine de regarder les trois cas.

© 2025 Mr. Lambrechts

Les nombres

La colonne `absences` contient des nombres entiers. Ils sont rangés selon leur valeur, comme tu t'y attends.

```sql
SELECT prenom, nom, absences
FROM eleve
ORDER BY absences DESC;
```

```
prenom  | nom       | absences
--------+-----------+---------
Louis   | Istas     | 12
Hugo    | Englebert | 8
Noah    | Charlier  | 5
Camille | Jacques   | 4
Nathan  | Gérard    | 3
Lucas   | Adam      | 2
Léa     | Delvaux   | 1
Arthur  | Kevers    | 1
Emma    | Bastin    | 0
Chloé   | Fontaine  | 0
Manon   | Hubert    | 0
Sarah   | Lambert   | 0
```

`12` arrive bien avant `8`. Ce n'est vrai que parce que la colonne contient de vrais nombres.

> ⚠️ Si les absences avaient été enregistrées comme du texte, SQLite les comparerait caractère par caractère, comme dans un dictionnaire. `12` serait alors rangé avant `2`, parce que le caractère `1` vient avant le caractère `2`. Un nombre doit donc être stocké comme un nombre.

Les dates

Les dates de naissance sont stockées comme du **texte**, au format `AAAA-MM-JJ` (année, mois, jour). SQLite les compare donc caractère par caractère, de gauche à droite.

Comme l'année est écrite en premier, puis le mois, puis le jour, l'ordre alphabétique tombe exactement sur l'ordre chronologique :

```
2009-12-11
2010-01-30   ← l'année 2010 est plus grande que 2009, le reste n'est même pas regardé
2010-02-27   ← même année : on compare le mois, 02 est plus grand que 01
```

> ⚠️ Ce tri ne fonctionne qu'avec le format `AAAA-MM-JJ`. Avec des dates écrites `JJ/MM/AAAA`, `01/09/2010` serait rangé avant `08/09/2009`, parce que la comparaison commencerait par le jour.

Le texte

Trie maintenant les élèves par commune.

© 2025 Mr. Lambrechts

```sql
SELECT nom, prenom, commune
FROM eleve
ORDER BY commune;
```

```
nom       | prenom  | commune
----------+---------+----------
Englebert | Hugo    | Hannut
Jacques   | Camille | Hannut
Adam      | Lucas   | Jodoigne
Delvaux   | Léa     | Jodoigne
Fontaine  | Chloé   | Jodoigne
Istas     | Louis   | Jodoigne
Charlier  | Noah    | Perwez
Hubert    | Manon   | Perwez
Kevers    | Arthur  | Ramillies
Bastin    | Emma    | Wavre
Gérard    | Nathan  | Wavre
Lambert   | Sarah   | jodoigne
```

Regarde la dernière ligne. Sarah Lambert habite Jodoigne, comme quatre autres élèves. Pourtant, elle n'est pas rangée avec eux : elle arrive après Wavre.

La raison se voit dans la colonne : sa commune a été encodée `jodoigne`, avec une minuscule. C'est une faute de frappe. Pour SQLite, `Jodoigne` et `jodoigne` sont deux textes différents.

Pour comparer du texte, SQLite ne consulte pas un dictionnaire. Chaque caractère est enregistré sous la forme d'un numéro, et SQLite compare ces numéros. Dans cette numérotation, **toutes les majuscules passent avant toutes les minuscules**. Le `j` minuscule vient donc après le `W` majuscule.

Les lettres accentuées ont un numéro encore plus grand. Trie les prénoms pour le constater :

```sql
SELECT prenom
FROM eleve
ORDER BY prenom;
```

```
prenom
-------
Arthur
Camille
Chloé
Emma
Hugo
Louis
Lucas
Léa
Manon
Nathan
Noah
Sarah
```

Dans un dictionnaire, Léa viendrait avant Louis. Ici, elle vient après Lucas : SQLite compare `L` avec `L`, puis `é` avec `o` et `u`, et le `é` a un numéro plus grand que toutes les lettres sans accent.

© 2025 Mr. Lambrechts

🧠 SQLite trie le texte caractère par caractère : les majuscules d'abord, puis les minuscules, puis les lettres accentuées. Une faute de majuscule suffit à envoyer une ligne au bout de la liste.

ℹ️ La faute sur `jodoigne` se corrige avec une requête de modification (`UPDATE`), qui n'est pas le sujet de cet article. Garde-la pour l'instant : elle montre bien comment SQL compare le texte.

Trier sur plusieurs colonnes

Trier par commune regroupe les élèves d'une même commune. Mais à l'intérieur de Jodoigne, dans quel ordre sont-ils ? Rien ne le garantit. Pour le décider, on ajoute une deuxième colonne après une virgule.

```sql
SELECT commune, nom, prenom
FROM eleve
ORDER BY commune, nom;
```

```
commune   | nom       | prenom
----------+-----------+--------
Hannut    | Englebert | Hugo
Hannut    | Jacques   | Camille
Jodoigne  | Adam      | Lucas
Jodoigne  | Delvaux   | Léa
Jodoigne  | Fontaine  | Chloé
Jodoigne  | Istas     | Louis
Perwez    | Charlier  | Noah
Perwez    | Hubert    | Manon
Ramillies | Kevers    | Arthur
Wavre     | Bastin    | Emma
Wavre     | Gérard    | Nathan
jodoigne  | Lambert   | Sarah
```

SQLite trie d'abord sur `commune`. La deuxième colonne, `nom`, ne sert que pour départager les lignes qui ont la même commune.

Chaque colonne peut avoir son propre sens de tri. Voici les élèves rangés par commune, et dans chaque commune du plus absent au moins absent :

```sql
SELECT commune, nom, absences
FROM eleve
ORDER BY commune, absences DESC;
```

```
commune   | nom       | absences
----------+-----------+---------
Hannut    | Englebert | 8
Hannut    | Jacques   | 4
```

© 2025 Mr. Lambrechts

```
Jodoigne  | Istas     | 12
Jodoigne  | Adam      | 2
Jodoigne  | Delvaux   | 1
Jodoigne  | Fontaine  | 0
Perwez    | Charlier  | 5
Perwez    | Hubert    | 0
Ramillies | Kevers    | 1
Wavre     | Gérard    | 3
Wavre     | Bastin    | 0
jodoigne  | Lambert   | 0
```

⚠️ `DESC` ne s'applique qu'à la colonne qu'il suit. Dans `ORDER BY commune, absences DESC`, les communes restent en ordre croissant ; seules les absences sont en ordre décroissant.

Sans ORDER BY, aucun ordre n'est promis

Au début de cet article, la requête sans tri affichait les élèves par ordre alphabétique. Il serait tentant de compter là-dessus. C'est une erreur.

Sans `ORDER BY`, SQLite renvoie les lignes dans l'ordre qui lui coûte le moins de travail. Aujourd'hui, c'est souvent l'ordre dans lequel les lignes ont été ajoutées. Après des suppressions, des ajouts, ou sur un autre logiciel de base de données, cet ordre peut changer sans prévenir.

🧠 Sans `ORDER BY`, personne ne te promet un ordre. Dès que l'ordre compte pour répondre à la question, écris un `ORDER BY`.

La place de ORDER BY dans la requête

Une requête SQL est faite de morceaux qui commencent chacun par un mot-clé : `SELECT …`, `FROM …`, `ORDER BY …`. Ces morceaux s'appellent des **clauses**, et ils doivent apparaître dans un ordre fixe.

`ORDER BY` se place **après** `FROM`, en fin de requête. Si tu l'écris ailleurs, SQLite refuse la requête :

```sql
SELECT nom, prenom
ORDER BY nom
FROM eleve;
```

```
near "FROM": syntax error
```

© 2025 Mr. Lambrechts

Le message se lit *erreur de syntaxe près de FROM* : la requête est mal construite, et SQLite s'est perdu en arrivant au mot `FROM`.

> 💡 Quand une requête contient d'autres clauses, comme `WHERE` qui filtre les lignes, `ORDER BY` vient après elles. Il ne reste que `LIMIT`, qui limite le nombre de lignes, pour se placer encore après.

🧪 Exercices

Pour chaque exercice, écris la requête dans l'onglet **Exécuter le SQL**, exécute-la, puis vérifie que le résultat répond vraiment à la question.

Exercice 1 — Premiers tris ★☆☆☆☆

Écris une requête qui affiche :

1. le prénom et le nom de tous les élèves, par ordre alphabétique de prénom ;
2. le nom et le nombre d'absences, du moins absent au plus absent ;
3. le prénom, le nom et la date de naissance, du plus jeune au plus âgé ;
4. toutes les colonnes, triées par nom de Z à A.

Exercice 2 — Prédire le résultat ★★☆☆☆

Sans exécuter les requêtes, écris sur papier la **première** et la **dernière** ligne du résultat. Exécute ensuite pour vérifier.

```sql
-- a)
SELECT prenom FROM eleve ORDER BY prenom DESC;

-- b)
SELECT nom, commune FROM eleve ORDER BY commune DESC;

-- c)
SELECT prenom, absences FROM eleve ORDER BY absences, prenom;
```

Pour la requête a), à quelle place arrive Léa ? Explique pourquoi.

Exercice 3 — Trier sur plusieurs colonnes ★★☆☆☆

1. Affiche le nom, la commune et la date de naissance des élèves, rangés par commune, puis du plus âgé au plus jeune dans chaque commune.
2. Affiche le nom et les absences, du plus absent au moins absent. En cas d'égalité, range les élèves par ordre alphabétique de nom.
3. Dans le résultat de la question 1, Sarah Lambert n'est pas rangée avec les autres élèves de Jodoigne. Explique pourquoi en une phrase.

© 2025 Mr. Lambrechts

Exercice 4 — Réparer des requêtes ★★★☆☆

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

```sql
-- a) Les élèves par ordre alphabétique de nom
SELECT nom, prenom FROM eleve ORDER nom;

-- b) Les élèves par ordre alphabétique de nom
SELECT nom, prenom ORDER BY nom FROM eleve;

-- c) Du plus absent au moins absent, puis par nom
SELECT nom, absences FROM eleve ORDER BY absences DESC nom;

-- d) Les élèves du plus âgé au plus jeune
SELECT nom FROM eleve ORDER BY date;

-- e) Les communes de A à Z, et dans chaque commune du plus absent au moins absent
SELECT nom, commune, absences FROM eleve ORDER BY commune DESC, absences;
```

> ⚠️ La requête e) s'exécute sans erreur. Elle est pourtant fausse. Une requête qui renvoie un mauvais résultat en silence est plus dangereuse qu'une requête qui refuse de fonctionner.

✍️ À retenir

- `ORDER BY colonne` (*ordonner par*) trie les lignes du résultat, sans modifier la table.
- Le tri est croissant par défaut (`ASC`) ; `DESC` le rend décroissant, pour la seule colonne qu'il suit.
- `ORDER BY a, b` trie sur `a`, puis sur `b` pour départager les égalités.
- Les nombres sont triés par valeur ; les dates `AAAA-MM-JJ` stockées en texte sont triées dans l'ordre chronologique.
- Le texte est trié caractère par caractère : majuscules, puis minuscules, puis lettres accentuées.
- Sans `ORDER BY`, aucun ordre n'est garanti.
- `ORDER BY` se place après `FROM` et `WHERE`, et avant `LIMIT`.

Suite

Trier toutes les lignes, c'est utile. Mais souvent, seules quelques-unes t'intéressent : les élèves de Wavre, ceux qui ont plus de quatre absences… C'est le rôle de Filtrer les lignes avec WHERE.

© 2025 Mr. Lambrechts

© 2025 Mr. Lambrechts