Les valeurs absentes : NULL

Trois élèves n'ont pas d'adresse email. Les retrouver semble simple, et pourtant WHERE email = NULL ne renvoie rien. Cet article explique ce qu'est NULL et comment le tester avec IS NULL.

4TTR 5TTR 6TTR Découverte

L'école veut envoyer un rappel aux familles des élèves qui n'ont pas encore communiqué d'adresse email. Il faut donc trouver les lignes où la colonne `email` est vide. La requête qui vient à l'esprit semble évidente. Elle ne renvoie pourtant rien du tout.

> 💡 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. expliquer ce que signifie `NULL` dans une base de données ;
2. distinguer `NULL`, le nombre `0` et le texte vide `''` ;
3. expliquer pourquoi `WHERE email = NULL` ne renvoie jamais rien ;
4. retrouver les valeurs absentes avec `IS NULL` et `IS NOT NULL` ;
5. repérer une condition qui écarte les `NULL` sans le vouloir ;
6. prévoir la place des `NULL` dans un tri.

Des cases sans valeur

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.

Affiche les adresses email :

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

```
prenom  | nom       | email
--------+-----------+------------------------
Lucas   | Adam      | lucas.adam@ecole.be
Emma    | Bastin    | emma.bastin@ecole.be
Noah    | Charlier  | NULL
Léa     | Delvaux   | lea.delvaux@ecole.be
Hugo    | Englebert | hugo.englebert@ecole.be
Chloé   | Fontaine  | NULL
Nathan  | Gérard    | nathan.gerard@ecole.be
Manon   | Hubert    | manon.hubert@ecole.be
Louis   | Istas     | louis.istas@ecole.be
Camille | Jacques   | NULL
Arthur  | Kevers    | arthur.kevers@ecole.be
Sarah   | Lambert   | sarah.lambert@ecole.be
```

Pour Noah, Chloé et Camille, la case affiche `NULL`. Ce n'est pas une adresse qui s'écrirait N-U-L-L. C'est une marque qui signifie : **ici, il n'y a pas de valeur**. L'école ne connaît pas leur adresse.

> 📖 **NOUVELLE NOTION : NULL**
>
> `NULL` est la marque qu'une base de données place dans une case quand la valeur est **absente ou inconnue**. Ce n'est pas une valeur comme les autres : ce n'est ni le nombre zéro, ni un texte vide. Le mot vient de l'anglais *null*, qui veut dire *nul, aucun*.
>
> En anglais : GB ***null value***.

> ℹ️ Toutes les colonnes n'acceptent pas `NULL`. Dans l'onglet **Structure de la base**, les colonnes `nom` et `prenom` portent la mention `NOT NULL` (*pas nul*) : la base refuse un élève sans nom. La colonne `email` n'a pas cette contrainte, elle peut donc rester vide.

NULL n'est ni zéro ni un texte vide

La colonne `absences` contient des zéros. Cherche-les :

```sql
SELECT prenom, nom, absences
FROM eleve
WHERE absences = 0;
```

```
prenom | nom      | absences
-------+----------+---------
Emma   | Bastin   | 0
Chloé  | Fontaine | 0
Manon  | Hubert   | 0
Sarah  | Lambert  | 0
```

Pour Emma, l'école sait quelque chose : elle n'a jamais été absente. Zéro est une information **connue**.

Cherche maintenant un email égal au texte vide, deux apostrophes collées :

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

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

Aucune ligne. Les trois cases vides ne contiennent pas un texte de zéro caractère : elles ne contiennent rien du tout.

Écriture	Signification	Exemple
`0`	un nombre connu, qui vaut zéro	Emma n'a aucune absence
`''`	un texte connu, qui ne contient aucun caractère	quelqu'un a encodé une adresse vide
`NULL`	aucune valeur : on ne sait pas	l'adresse de Noah n'a jamais été communiquée

Le piège de WHERE email = NULL

Pour trouver les élèves sans adresse, la requête naturelle est celle-ci :

```sql
SELECT prenom, nom, email
FROM eleve
WHERE email = NULL;
```

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

Aucune ligne, et aucun message d'erreur. La requête est valide, mais elle ne renverra **jamais** rien, quelle que soit la table.

Pour comprendre, pose la question à voix haute pour Noah : « est-ce que son adresse, qu'on ne connaît pas, est égale à une valeur qu'on ne connaît pas ? ». La seule réponse honnête est : **on ne sait pas**. Et c'est exactement ce que répond SQLite.

Tu peux le vérifier. Exécute ces trois comparaisons, sans table :

```sql
SELECT NULL = NULL AS test1,
       'a' = 'a' AS test2,
       NULL <> 'a' AS test3;
```

```
test1 | test2 | test3
------+-------+------
NULL  | 1     | NULL
```

SQLite affiche `1` pour *vrai* et `0` pour *faux*. La comparaison `'a' = 'a'` est vraie. Mais toute comparaison avec `NULL` donne `NULL` : ni vrai, ni faux, **inconnu**.

Or `WHERE` ne garde que les lignes pour lesquelles la condition est **vraie**. Une condition inconnue n'est pas vraie, donc la ligne est écartée.

> 🧠 Toute comparaison avec `NULL` (`=` , `<>` , `<` , `>` , `LIKE` …) donne un résultat inconnu. `WHERE` écarte les lignes dont la condition est inconnue. C'est pour cela que `WHERE email = NULL` ne renvoie jamais rien.

Tester l'absence de valeur avec IS NULL

Puisque `=` ne peut pas tester `NULL`, SQL propose un opérateur spécial : `IS NULL`, qui veut dire *est nul*.

```sql
SELECT prenom, nom, email
FROM eleve
WHERE email IS NULL;
```

```
prenom  | nom      | email
--------+----------+------
Noah    | Charlier | NULL
Chloé   | Fontaine | NULL
Camille | Jacques  | NULL
```

Les trois élèves sont enfin là. `IS NULL` ne compare pas deux valeurs : il pose une autre question, « cette case est-elle vide ? », à laquelle la réponse est toujours oui ou non.

> 📖 **NOUVELLE NOTION : IS NULL ET IS NOT NULL**
>
> `IS NULL` (*est nul*) est vrai quand la case ne contient aucune valeur. `IS NOT NULL` (*n'est pas nul*) est vrai quand la case contient une valeur, quelle qu'elle soit. Ce sont les seuls moyens fiables de tester l'absence de valeur : on n'écrit jamais `= NULL` .
>
> En anglais : GB ***IS NULL***, littéralement *is null*.

L'inverse, `IS NOT NULL`, retrouve les élèves qui ont une adresse :

```sql
SELECT prenom, nom, email
FROM eleve
WHERE email IS NOT NULL;
```

```
prenom | nom       | email
-------+-----------+-----------------------
Lucas  | Adam      | lucas.adam@ecole.be
Emma   | Bastin    | emma.bastin@ecole.be
Léa    | Delvaux   | lea.delvaux@ecole.be
Hugo   | Englebert | hugo.englebert@ecole.be
```

```
Nathan | Gérard    | nathan.gerard@ecole.be
Manon  | Hubert    | manon.hubert@ecole.be
Louis  | Istas     | louis.istas@ecole.be
Arthur | Kevers    | arthur.kevers@ecole.be
Sarah  | Lambert   | sarah.lambert@ecole.be
```

`IS NULL` se combine avec d'autres conditions, comme n'importe quelle condition. Voici les élèves sans adresse qui ont plus de 3 absences, et dont il faudrait contacter la famille autrement :

```sql
SELECT prenom, nom, email
FROM eleve
WHERE email IS NULL
  AND absences > 3;
```

```
prenom  | nom      | email
--------+----------+------
Noah    | Charlier | NULL
Camille | Jacques  | NULL
```

Les NULL disparaissent aussi des conditions « différent de »

Le piège de `NULL` ne se limite pas à `= NULL`. Cherche tous les élèves sauf Lucas Adam, en te servant de son adresse :

```sql
SELECT prenom, nom, email
FROM eleve
WHERE email <> 'lucas.adam@ecole.be';
```

```
prenom | nom       | email
-------+-----------+------------------------
Emma   | Bastin    | emma.bastin@ecole.be
Léa    | Delvaux   | lea.delvaux@ecole.be
Hugo   | Englebert | hugo.englebert@ecole.be
Nathan | Gérard    | nathan.gerard@ecole.be
Manon  | Hubert    | manon.hubert@ecole.be
Louis  | Istas     | louis.istas@ecole.be
Arthur | Kevers    | arthur.kevers@ecole.be
Sarah  | Lambert   | sarah.lambert@ecole.be
```

Huit lignes au lieu de onze. Noah, Chloé et Camille ont disparu. Leur adresse n'est pas celle de Lucas, bien sûr, mais SQLite ne peut pas l'affirmer : comparer `NULL` à une adresse donne un résultat inconnu, et la ligne est écartée.

Pour les récupérer, il faut les demander explicitement :

```sql
SELECT prenom, nom, email
FROM eleve
```

```sql
WHERE email <> 'lucas.adam@ecole.be'
   OR email IS NULL;
```

```
prenom  | nom       | email
--------+-----------+-----------------------
Emma    | Bastin    | emma.bastin@ecole.be
Noah    | Charlier  | NULL
Léa     | Delvaux   | lea.delvaux@ecole.be
Hugo    | Englebert | hugo.englebert@ecole.be
Chloé   | Fontaine  | NULL
Nathan  | Gérard    | nathan.gerard@ecole.be
Manon   | Hubert    | manon.hubert@ecole.be
Louis   | Istas     | louis.istas@ecole.be
Camille | Jacques   | NULL
Arthur  | Kevers    | arthur.kevers@ecole.be
Sarah   | Lambert   | sarah.lambert@ecole.be
```

⚠️ Une condition sur une colonne qui peut contenir `NULL` écarte ces lignes en silence, même avec `<>` ou `NOT LIKE`. Pose-toi toujours la question : « et les cases vides, je les veux ou pas ? ».

La place des NULL dans un tri

Trie les élèves par adresse email :

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

```
prenom  | nom       | email
--------+-----------+-----------------------
Noah    | Charlier  | NULL
Chloé   | Fontaine  | NULL
Camille | Jacques   | NULL
Arthur  | Kevers    | arthur.kevers@ecole.be
Emma    | Bastin    | emma.bastin@ecole.be
Hugo    | Englebert | hugo.englebert@ecole.be
Léa     | Delvaux   | lea.delvaux@ecole.be
Louis   | Istas     | louis.istas@ecole.be
Lucas   | Adam      | lucas.adam@ecole.be
Manon   | Hubert    | manon.hubert@ecole.be
Nathan  | Gérard    | nathan.gerard@ecole.be
Sarah   | Lambert   | sarah.lambert@ecole.be
```

Dans un tri croissant, SQLite place les `NULL` **en premier**, comme s'ils étaient plus petits que toutes les valeurs. Avec `ORDER BY email DESC`, ils passent en dernier.

ℹ️ Tous les logiciels ne font pas ce choix. PostgreSQL, par exemple, place les `NULL` à la fin d'un tri croissant. Si leur place compte, vérifie-la.

🧪 Exercices

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

Exercice 1 — Retrouver les cases vides ★☆☆☆☆

Écris une requête qui affiche le prénom et le nom :

1. des élèves sans adresse email, triés par nom ;
2. des élèves qui ont une adresse email ;
3. des élèves qui ont une adresse email et aucune absence ;
4. des élèves dont le nombre d'absences est inconnu. Combien en trouves-tu ?

Exercice 2 — Zéro, vide ou inconnu ? ★★☆☆☆

Pour chaque situation, indique ce qu'il faudrait enregistrer dans la case : `0`, `''` ou `NULL`. Justifie en une phrase.

1. Un élève n'a manqué aucun cours.
2. Un nouvel élève vient d'arriver ; on ne sait pas encore combien d'absences il a eues dans son ancienne école.
3. Une famille a déclaré ne pas avoir d'adresse email.
4. Un élève a oublié de remplir la case « commune » sur son formulaire.

Exercice 3 — Prédire le résultat ★★☆☆☆

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

```sql
-- a)
SELECT nom FROM eleve WHERE email = NULL;

-- b)
SELECT nom FROM eleve WHERE email IS NULL;

-- c)
SELECT nom FROM eleve WHERE email <> 'emma.bastin@ecole.be';

-- d)
SELECT nom FROM eleve WHERE email NOT LIKE '%.a%';

-- e)
SELECT nom FROM eleve WHERE email = 'NULL';
```

Exercice 4 — Réparer des requêtes ★★★☆☆

Chacune de ces requêtes 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 sans adresse email
SELECT prenom, nom FROM eleve WHERE email = NULL;

-- b) Les élèves sans adresse email
SELECT prenom, nom FROM eleve WHERE email = 'NULL';

-- c) Tous les élèves sauf Lucas Adam (il y en a onze)
SELECT prenom, nom FROM eleve WHERE email <> 'lucas.adam@ecole.be';

-- d) Les élèves qui ont une adresse email
SELECT prenom, nom FROM eleve WHERE email <> NULL;
```

✍️ À retenir

- `NULL` marque une valeur absente ou inconnue. Ce n'est ni `0`, ni le texte vide `''`.
- Toute comparaison avec `NULL` donne un résultat inconnu, et `WHERE` écarte les lignes dont la condition est inconnue.
- `WHERE email = NULL` ne renvoie jamais rien : on écrit `WHERE email IS NULL`.
- `IS NOT NULL` retrouve les cases qui contiennent une valeur.
- Une condition comme `email <> '…'` écarte aussi les `NULL` : ajoute `OR email IS NULL` si tu les veux.
- Dans SQLite, les `NULL` arrivent en premier dans un tri croissant.

Suite

Tu sais maintenant filtrer, trier et gérer les cases vides. Pour ne garder que les premières lignes d'un résultat, comme « les trois élèves les plus absents », passe à Limiter le nombre de lignes avec LIMIT.