Fiche à étudier — Filtrer et analyser ses données

Tout ce qu'il faut connaître par cœur pour interroger une table : vocabulaire, syntaxe, ordre des clauses, pièges et questions d'auto-évaluation.

4TTR 5TTR 6TTR Synthèse

Cette fiche rassemble ce qu'il faut savoir par cœur pour trier, filtrer, compter et regrouper les lignes d'une table ; les exemples portent sur la table `eleve` (colonnes `id`, `nom`, `prenom`, `date_naissance`, `commune`, `email`, `absences`).

Le vocabulaire

Terme	En anglais	Définition
requête	query	Question posée à la base de données, écrite en langage SQL.
clause	clause	Morceau de requête qui commence par un mot-clé (`SELECT`, `FROM`, `WHERE`…). Les clauses présentes d
trier (`ORDER BY`)	sort	Ranger les lignes d'un résultat selon les valeurs d'une ou plusieurs colonnes. Le tri change l'ordre d'affich
condition	condition	Affirmation vérifiée sur chaque ligne, vraie ou fausse. `WHERE` ne garde que les lignes pour lesquelles elle
opérateur de comparaison	comparison operator	Symbole qui compare deux valeurs : `=`, `<>`, `<`, `>`, `<=`, `>=`.
priorité	operator precedence	Ordre dans lequel les opérateurs sont évalués. `AND` est évalué avant `OR` ; les parenthèses imposent un
motif	pattern	Modèle de texte qui décrit une famille de valeurs. `LIKE` garde les valeurs qui correspondent au motif.
joker	wildcard	Caractère qui, dans un motif, remplace d'autres caractères : `%` pour n'importe quelle suite (même vide),
NULL	null value	Marque d'une valeur absente ou inconnue. Ce n'est ni `0`, ni le texte vide `''`.
IS NULL / IS NOT NULL	IS NULL	Tests vrais quand la case est vide / contient une valeur. Seuls moyens fiables de tester `NULL`.
LIMIT	limit	Ne garde que les `n` premières lignes du résultat, après le filtre et le tri.
DISTINCT / doublon	distinct / duplicate	`DISTINCT` affiche chaque ligne du résultat une seule fois ; deux lignes identiques sont des doublons.
fonction d'agrégation	aggregate function	Fonction qui calcule une seule valeur à partir d'un ensemble de lignes : `COUNT`, `SUM`, `AVG`, `MIN`, `MAX`.
regroupement (`GROUP BY`)	group by	Range les lignes en groupes de même valeur ; les agrégats sont calculés pour chaque groupe, une ligne

La syntaxe

SELECT et FROM — lire une table

`SELECT` = *sélectionne* (les colonnes), `FROM` = *depuis* (la table). `AS` = *en tant que* : renomme une colonne dans le résultat.

```sql
SELECT prenom, nom AS nom_de_famille
FROM eleve;
```

ORDER BY — trier

Ordonner par. `ASC` = croissant (par défaut), `DESC` = décroissant, pour la seule colonne qu'il suit.

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

WHERE — filtrer

Où. Texte et dates entre apostrophes, nombres sans.

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

```sql
WHERE absences > 4;

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

AND, OR et parenthèses

`AND` = *et* (toutes vraies), `OR` = *ou* (au moins une vraie).

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

BETWEEN — intervalle

Entre. Les deux bornes sont comprises ; la plus petite s'écrit en premier.

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

IN et NOT IN — liste

Dans / *pas dans*.

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

LIKE et NOT LIKE — motif

Comme. `%` = n'importe quelle suite de caractères, `_` = exactement un caractère.

```sql
SELECT prenom, nom
FROM eleve
WHERE nom LIKE 'D%';        -- commence par D

SELECT prenom, email
FROM eleve
WHERE email LIKE '%@ecole.be';   -- se termine par @ecole.be
```

IS NULL et IS NOT NULL — valeur absente

Est nul / *n'est pas nul*.

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

LIMIT — nombre de lignes

Limite. Toujours avec un `ORDER BY`. `OFFSET n` (*décalage*) saute les `n` premières lignes.

```sql
SELECT prenom, nom, absences
FROM eleve
ORDER BY absences DESC
LIMIT 3;
```

DISTINCT — sans répétition

Juste après `SELECT`. S'applique à toutes les colonnes du `SELECT`.

```sql
SELECT DISTINCT commune
FROM eleve
ORDER BY commune;
```

Fonctions d'agrégation

`COUNT` = *compter*, `SUM` = *somme*, `AVG` = *moyenne*, `MIN` / `MAX`, `ROUND(valeur, n)` = *arrondir* à `n` décimales.

```sql
SELECT COUNT(*) AS nb_eleves,
       COUNT(email) AS nb_emails,
       COUNT(DISTINCT commune) AS nb_communes,
       SUM(absences) AS total,
       ROUND(AVG(absences), 1) AS moyenne,
       MIN(date_naissance) AS plus_ancienne
FROM eleve;
```

GROUP BY — un résultat par groupe

Regrouper par. Le `SELECT` ne contient que les colonnes regroupées et des agrégats.

```sql
SELECT commune, COUNT(*) AS nb_eleves
FROM eleve
GROUP BY commune
ORDER BY nb_eleves DESC, commune;
```

L'ordre des clauses

Les clauses sont facultatives, sauf `SELECT` (et `FROM` pour lire une table). Celles qui sont présentes apparaissent **toujours** dans cet ordre :

```sql
SELECT ...      -- 1. Qu'est-ce que je veux voir ?
FROM ...        -- 2. Où sont les données ?
WHERE ...       -- 3. Quelles lignes ?
GROUP BY ...    -- 4. Regroupées comment ?
ORDER BY ...    -- 5. Dans quel ordre ?
LIMIT ...;      -- 6. Combien ?
```

Une clause mal placée provoque l'erreur `near "…": syntax error`, où `…` est le mot où SQLite s'est perdu.

Les pièges à connaître

Piège	Ce qui se passe	La bonne é
`WHERE email = NULL`	Aucune ligne, jamais, sans erreur : une comparaison avec `NULL` est inconnue, pas vraie.	`WHERE ema`
`WHERE email <> 'x@ecole.be'`	Les lignes `NULL` disparaissent aussi.	ajouter `OR`
`WHERE a = 1 OR b = 2 AND c = 3`	`AND` est évalué d'abord : `a = 1 OR (b = 2 AND c = 3)`.	parenthèse
`WHERE prenom = Emma`	Erreur `no such column: Emma` : sans apostrophes, c'est un nom de colonne.	`WHERE pre`
`WHERE date_naissance > 2010-01-01`	Aucune erreur mais résultat faux : `2010-01-01` est le calcul 2008.	`WHERE dat`
`COUNT(email)` pour compter des élèves	Les `NULL` ne sont pas comptés : 9 au lieu de 12.	`COUNT(*)`
`LIMIT 3` sans `ORDER BY`	« Les trois premiers » selon un ordre non garanti.	`ORDER BY`
`WHERE commune = 'Jodoigne'`	La ligne encodée `jodoigne` est ratée : `=` fait la différence entre majuscules et minuscules.	`LIKE 'jod`
`LIKE 'GÉRARD'`	`LIKE` ignore la casse des lettres sans accent seulement : `É` et `é` restent différents.	respecter le
Tri du texte	Majuscules, puis minuscules, puis lettres accentuées : `jodoigne` après `Wavre`, `Léa` après `Lucas`.	écrire les d
`WHERE COUNT(*) >= 2`	Erreur `misuse of aggregate` : `WHERE` agit avant le calcul.	`HAVING CO`

✅ Teste-toi

Réponds sur papier, sans regarder la fiche, puis vérifie avec les réponses.

1. Que signifient littéralement `WHERE`, `ORDER BY` et `GROUP BY` ?
2. Dans quel ordre écris-tu les clauses `LIMIT`, `FROM`, `ORDER BY`, `SELECT`, `WHERE`, `GROUP BY` ?

3. Quel est le sens de tri par défaut de `ORDER BY` ? Comment l'inverser ?
4. Pourquoi `WHERE email = NULL` ne renvoie-t-il jamais rien ? Que faut-il écrire ?
5. Quelle est la différence entre `NULL`, `0` et `''` ?
6. Que renvoie `WHERE date_naissance > 2010-01-01`, sans apostrophes ? Pourquoi ?
7. Dans `WHERE commune = 'Perwez' OR commune = 'Hannut' AND absences > 3`, quelle partie est évaluée en premier ? Réécris la condition pour « Perwez ou Hannut, et plus de 3 absences ».
8. `BETWEEN 1 AND 4` garde-t-il les valeurs 1 et 4 ?
9. Que représentent les jokers `%` et `_` ? Écris un motif pour « nom de cinq lettres qui commence par L ».
10. Dans SQLite, `WHERE commune LIKE 'jodoigne'` trouve-t-il `Jodoigne` ? Et `WHERE nom LIKE 'GÉRARD'` trouve-t-il `Gérard` ?
11. Pourquoi faut-il presque toujours un `ORDER BY` avec `LIMIT` ?
12. Dans la table `eleve`, `COUNT(*)` vaut 12 et `COUNT(email)` vaut 9. Explique l'écart.
13. `SELECT DISTINCT commune FROM eleve` renvoie six lignes pour cinq communes réelles. Que peut-on en conclure ?
14. Pourquoi `WHERE COUNT(*) >= 2` provoque-t-il une erreur ?
15. Dans une requête avec `GROUP BY commune`, pourquoi ne faut-il pas ajouter `nom` dans le `SELECT` ?

▶ Réponses