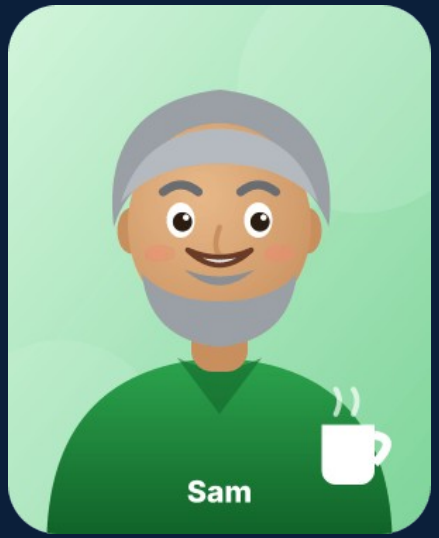
HEF-070 · MISSION 03 / 10

SQL de lecture : interroger les données

SELECT, WHERE, ORDER BY : trois mots et tu peux déjà poser la plupart de tes questions.





Badge : Interrogateur de données

80 min · intermédiaire

L'histoire

Maya veut des chiffres : « Quels membres de Lyon ont terminé plus de 3 missions ? Quels sont les 5 meilleurs scores ? Combien de nouveaux inscrits par mois ? » Kodi sait stocker, il doit maintenant interroger.



« SQL se lit presque comme une phrase, dit Sam. Tu décris ce que tu veux, la base décide comment le trouver. »

Tu apprends à écrire des requêtes de lecture claires et correctes.

Ta mission

À la fin, je serai capable de…

1. écrire une requête SELECT avec WHERE, ORDER BY, LIMIT
2. utiliser DISTINCT, LIKE, IN, BETWEEN
3. comprendre NULL et la logique à trois valeurs
4. utiliser des fonctions et expressions (COUNT, CASE, dates)
5. lire l'ordre logique d'exécution d'une requête

Défi d'échauffement

Défi d'échauffement

Voici une liste de membres avec ville et score. Écris en français **trois questions** que Maya pourrait poser (un filtre, un tri, un comptage). Pour chacune, décompose : *quelles colonnes* je veux, *quelles lignes* je garde, *dans quel ordre* je les affiche.

Le concept

Structure d'une requête

```
SELECT colonnes
FROM table
WHERE condition
ORDER BY colonnes
LIMIT n;
```

Clause	Rôle
`SELECT`	quelles colonnes (ou expressions)
`FROM`	de quelle table
`WHERE`	quelles **lignes** garder
`ORDER BY`	dans quel ordre (`ASC` / `DESC`)
`LIMIT`	combien au maximum

Le concept

Les conditions

```
WHERE ville = 'Lyon' AND score >= 50
WHERE ville IN ('Lyon', 'Lille')
WHERE score BETWEEN 10 AND 20
WHERE nom LIKE 'A%'          -- commence par A ; _ = un caractère
WHERE NOT (ville = 'Nice')
```

DISTINCT

`SELECT DISTINCT ville FROM membres` supprime les doublons.

NULL : l'absence de valeur

`NULL` signifie « inconnu ». Il se comporte **différemment** :

Le concept

NULL : l'absence de valeur (suite)

- `NULL = NULL` n'est ni vrai ni faux, c'est **inconnu**.
- On teste avec `IS NULL` / `IS NOT NULL`.
- Un `WHERE` ne garde que les lignes dont la condition est **vraie** (pas « inconnue »).

Expression	Résultat
`5 = NULL`	inconnu (ligne non gardée)
`NULL IS NULL`	vrai
`NULL + 1`	NULL
`COUNT(*)` vs `COUNT(colonne)`	`COUNT(colonne)` ignore les NULL

Expressions et fonctions

Le concept

Expressions et fonctions (suite)

```
SELECT nom, score * 1.1 AS score_bonus FROM membres;
SELECT UPPER(nom), LENGTH(nom) FROM membres;
SELECT nom,
       CASE WHEN score >= 80 THEN 'expert'
            WHEN score >= 40 THEN 'confirmé'
            ELSE 'débutant' END AS niveau
FROM membres;
```

L'ordre logique d'exécution

On écrit `SELECT` en premier, mais la base l'**évalue** dans cet ordre :



Le concept

L'ordre logique d'exécution (suite)

Description du schéma : l'ordre d'évaluation est FROM, WHERE, GROUP BY, HAVING, SELECT, ORDER BY, LIMIT. C'est pourquoi on ne peut pas utiliser un alias défini dans SELECT à l'intérieur du WHERE, mais on peut le faire dans ORDER BY.

Requêtes paramétrées

Ne **jamais** construire une requête en collant du texte venant de l'utilisateur : risque d'**injection SQL**. On utilise des **paramètres** :

```
bd.execute("SELECT * FROM membres WHERE ville = ?", (ville_saisie,))
```

Exemple pas à pas

Exemple pas à pas

Problème : répondre aux questions de Maya avec une base d'exemple.

```
import sqlite3

bd = sqlite3.connect(":memory:")
bd.executescript('''
CREATE TABLE membres (id INTEGER PRIMARY KEY, nom TEXT, ville TEXT, score INTEGER,
    inscrit_le TEXT);
INSERT INTO membres (nom, ville, score, inscrit_le) VALUES
 ('Ada','Lyon',90,'2025-01-10'), ('Bob','Lille',45,'2025-01-22'), ('Cléo','Lyon',6
   0,'2025-02-03'),
 ('Dan','Nice',NULL,'2025-02-14'), ('Eve','Lyon',30,'2025-02-20'), ('Félix','Lille
   ',75,'2025-03-01');
''')

def q(sql, params=()):
    return bd.execute(sql, params).fetchall()
```

Exemple pas à pas (suite)

```
print(q("SELECT nom, score FROM membres WHERE ville = 'Lyon' AND score > 40 ORDER
   BY score DESC"))
# [('Ada', 90), ('Cléo', 60)]
print(q("SELECT nom FROM membres ORDER BY score DESC LIMIT 3"))
# en SQLite, NULL est considéré plus petit que toute valeur : avec DESC il arrive
   en dernier
print(q("SELECT DISTINCT ville FROM membres ORDER BY ville"))
print(q("SELECT nom FROM membres WHERE score IS NULL"))                # [('Dan',
   )]
print(q("SELECT nom FROM membres WHERE score != 60"))                  # Dan abse
   nt !
print(q("SELECT substr(inscrit_le, 1, 7) AS mois, COUNT(*) FROM membres GROUP BY m
   ois ORDER BY mois"))
print(q('''SELECT nom, CASE WHEN score >= 70 THEN 'expert' WHEN score >= 40 THEN '
   confirmé' ELSE 'débutant' END
           FROM membres WHERE score IS NOT NULL ORDER BY score DESC'''))
```

Exemple pas à pas

Exemple pas à pas (suite)

Piège NULL : `WHERE score != 60` **ne renvoie pas Dan**, dont le score est NULL (la condition est « inconnue »). Pour l'inclure : `WHERE score != 60 OR score IS NULL`.

Injection SQL :

```
saisie = "Lyon' OR '1'='1"
# DANGEREUX : f"SELECT nom FROM membres WHERE ville = '{saisie}'" renverrait tous
   les membres
print(q("SELECT nom FROM membres WHERE ville = ?", (saisie,)))     # [] : traité c
   omme du texte
```

À toi, avec aide

Avec la table `membres` de l'exemple, écris les requêtes :

1. les noms des membres de Lyon ou de Lille dont le score dépasse 40 ;
2. les trois meilleurs scores, sans valeur NULL ;
3. les membres dont le nom contient la lettre « e » (minuscule ou majuscule) ;
4. le nombre de membres par ville, trié du plus grand au plus petit.

À toi, avec aide · réponse

Indices progressifs

Indice 1

Pour 1, utilise `IN ('Lyon', 'Lille')`.

Indice 2

Pour 2, ajoute `WHERE score IS NOT NULL` avant `ORDER BY ... LIMIT 3`.

Indice 3

Pour 3, `LIKE '%e%'` (SQLite ignore la casse pour l'ASCII par défaut).

À toi, avec aide · réponse

Réponse

```
import sqlite3

bd = sqlite3.connect(":memory:")
bd.executescript('''
CREATE TABLE membres (id INTEGER PRIMARY KEY, nom TEXT, ville TEXT, score INTEGER,
    inscrit_le TEXT);
INSERT INTO membres (nom, ville, score, inscrit_le) VALUES
 ('Ada','Lyon',90,'2025-01-10'), ('Bob','Lille',45,'2025-01-22'), ('Cléo','Lyon',6
   0,'2025-02-03'),
 ('Dan','Nice',NULL,'2025-02-14'), ('Eve','Lyon',30,'2025-02-20'), ('Félix','Lille
   ',75,'2025-03-01');
''')
print(bd.execute("SELECT nom FROM membres WHERE ville IN ('Lyon','Lille') AND scor
   e > 40").fetchall())
print(bd.execute("SELECT nom, score FROM membres WHERE score IS NOT NULL ORDER BY
   score DESC LIMIT 3").fetchall())
print(bd.execute("SELECT nom FROM membres WHERE nom LIKE '%e%'").fetchall())
print(bd.execute("SELECT ville, COUNT(*) AS n FROM membres GROUP BY ville ORDER BY
    n DESC, ville").fetchall())
```

À toi, en solo

Questions pour la direction. Avec une table `ventes(id, client, produit, quantite, prix_centimes, date, remise)` (la remise peut être NULL), écris :

1. le chiffre d'affaires de chaque produit (prix × quantité, avec remise éventuelle en pourcentage) ;
2. les 5 clients qui ont dépensé le plus ;
3. les ventes d'un mois donné (paramètre) ;
4. les ventes sans remise (remise NULL **ou** 0) ;
5. une colonne « gros panier » (oui si quantité ≥ 10).

Crée la table et 12 lignes d'exemple pour tester. Utilise des paramètres pour le mois.

À toi, en solo · réponse

Pistes de correction

```
import sqlite3

bd = sqlite3.connect(":memory:")
bd.executescript('''
CREATE TABLE ventes (id INTEGER PRIMARY KEY, client TEXT, produit TEXT, quantite I
   NTEGER, prix_centimes INTEGER, date TEXT, remise INTEGER);
INSERT INTO ventes (client, produit, quantite, prix_centimes, date, remise) VALUES
 ('Ada','Stylo',10,150,'2025-01-05',NULL), ('Ada','Cahier',2,300,'2025-01-07',10),
    ('Bob','Stylo',5,150,'2025-01-09',0),
 ('Bob','Sac',1,2500,'2025-02-02',20), ('Cléo','Cahier',12,300,'2025-02-04',NULL),
    ('Cléo','Règle',3,200,'2025-02-10',NULL),
 ('Dan','Stylo',1,150,'2025-02-11',5), ('Dan','Sac',2,2500,'2025-03-01',NULL), ('E
   ve','Règle',8,200,'2025-03-03',0),
 ('Eve','Cahier',1,300,'2025-03-04',NULL), ('Félix','Stylo',20,150,'2025-03-05',10
   ), ('Félix','Sac',1,2500,'2025-03-06',NULL);
''')
montant = "prix_centimes * quantite * (100 - COALESCE(remise, 0)) / 100.0"
print(bd.execute(f"SELECT produit, SUM({montant}) AS ca FROM ventes GROUP BY produ
   it ORDER BY ca DESC").fetchall())
```

Pistes de correction (suite)

```
print(bd.execute(f"SELECT client, SUM({montant}) AS depense FROM ventes GROUP BY c
   lient ORDER BY depense DESC LIMIT 5").fetchall())
print(bd.execute("SELECT id, client FROM ventes WHERE substr(date, 1, 7) = ?", ("2
   025-02",)).fetchall())
print(bd.execute("SELECT id FROM ventes WHERE remise IS NULL OR remise = 0").fetch
   all())
print(bd.execute("SELECT id, CASE WHEN quantite >= 10 THEN 'oui' ELSE 'non' END AS
    gros_panier FROM ventes").fetchall())
```

`COALESCE(remise, 0)` remplace NULL par 0 : indispensable pour que le calcul ne donne pas NULL.

Pièges classiques

Piège	Pourquoi c'est un problème	Comment l'éviter
`= NULL` au lieu de `IS NULL`	Aucune ligne renvoyée	`IS NULL` / `IS NOT NULL`
Construire du SQL avec du texte collé	Injection SQL	Requêtes paramétrées
`SELECT *` partout	Colonnes inutiles, fragile aux changements	Nommer les colonnes
Oublier `ORDER BY` avec `LIMIT`	Ordre arbitraire des lignes	Toujours trier avant de limiter

Mini-boss

Mini-boss

Tableau de bord SQL. À partir d'une base de 50 membres générée par un script (villes, scores, dates d'inscription, avec quelques NULL), écris et teste dix requêtes qui répondent à des questions de la direction :

1. répartition des membres par ville, avec le pourcentage ;
2. membres inscrits depuis moins de 30 jours (date fixe de référence) ;
3. médiane des scores (indice : trier et prendre le milieu avec `LIMIT` / `OFFSET`) ;
4. membres dont le score est inconnu ;
5. classement avec rang (sans fonctions de fenêtre, puis avec) ;
6. pagination : page 3 de 10 résultats ;
7. recherche de nom insensible à la casse avec préfixe fourni en paramètre ;
8. top 3 par ville (indice : sous-requête corrélée ou fonction de fenêtre) ;

Mini-boss (suite)

9. une requête volontairement vulnérable à l'injection, la démonstration de l'attaque, puis sa version corrigée ;
10. un tableau récapitulatif imprimé proprement.

Mini-boss · réponse

Pistes de correction

Un bon travail génère les données avec `random.Random(graine)`, vérifie chaque requête avec une version Python équivalente sur les mêmes données (comparaison des résultats), utilise `COALESCE` et `IS NULL` correctement, des paramètres partout où une valeur vient de l'extérieur, `ROW_NUMBER() OVER (PARTITION BY ville ORDER BY score DESC)` pour le top 3 par ville (SQLite ≥ 3.25), `LIMIT 10 OFFSET 20` pour la page 3, et une démonstration claire de l'injection (`' OR '1'='1`) suivie de la version paramétrée qui ne renvoie rien.

Quiz éclair

Quiz éclair

1. Pour tester l'absence de valeur on écrit :

A. = NULL
B. IS NULL
C. == NULL

2. Quelle clause filtre les lignes ?

A. ORDER BY
B. LIMIT
C. WHERE

3. Pour éviter l'injection SQL, on utilise :

A. des requêtes paramétrées

Quiz éclair

Quiz éclair (suite)

B. des guillemets doubles
C. des majuscules

4. 🔁 *Révision mission 02* : La 3FN élimine les dépendances :

A. vers la clé primaire
B. transitives
C. entre deux tables

5. 🔁 *Révision mission 02* : Une anomalie de modification se produit quand :

A. un index est manquant
B. la base est trop grande
C. une même information est stockée à plusieurs endroits

Quiz éclair · réponse

Réponses

1. **B**.
2. **C**.
3. **A**.
4. **B**.
5. **C**.

Je réfléchis

1. Quelle question de ta vie professionnelle ou personnelle voudrais-tu poser à des données ?
2. Pourquoi NULL est-il source de tant d'erreurs ?

À retenir

1. SELECT, FROM, WHERE, ORDER BY, LIMIT.
2. NULL = inconnu : IS NULL, COALESCE.
3. Ordre logique : FROM, WHERE, GROUP BY, HAVING, SELECT, ORDER BY.
4. CASE pour des colonnes calculées.
5. Requêtes paramétrées contre l'injection.

MISSION ACCOMPLIE

+100 XP

Badge débloqué : Interrogateur de données

Prochaine mission : SQL d'écriture : modifier les données

