Support de cours

SQL de base

–Organisme de formation professionnelle, 2-4 rue Vincent Van Gogh - 93360 Neuilly-Plaisance

Email : contact@qualis-academy.fr | Site web : www.qualis-academy.fr

Table des matières

1. INTRODUCTION GENERALE AU SQL

1.1 Généralités sur le SQL

SQL signifie **Structured Query Language** (Langage de Requête Structuré). C'est un langage standardisé qui permet de communiquer avec une base de données relationnelle.

Une base de données est un ensemble organisé d'informations stockées dans des tables. Chaque table contient :

Des colonnes (appelées champs).

Des lignes (appelées enregistrements).

	language_id	name	last_update
▶	1	English	15/02/2006 05:02
	2	Italian	15/02/2006 05:02
	3	Japanese	15/02/2006 05:02
	4	Mandarin	15/02/2006 05:02
	5	French	15/02/2006 05:02
	6	German	15/02/2006 05:02
	7	Arabic	16/06/2025 09:19
	8	Portugais	16/06/2025 09:19
	9	espagnol	17/06/2025 09:19

1.2 Pourquoi un testeur logiciel a besoin de connaître les bases du SQL

En tant que testeur logiciel, vous serez amené à :

- Vérifier que les données sont correctement enregistrées dans la base après une action utilisateur
- Créer des jeux de données de test pour différents scénarios
- Valider l'intégrité et la cohérence des données
- Identifier rapidement la source d'un dysfonctionnement (interface ou base de données)
- Etc...

Le SQL est donc un outil indispensable pour un testeur qui souhaite être autonome et efficace.

1.3 Qu'est-ce que le SQL

Le SQL est un langage qui permet d'effectuer quatre grandes catégories d'opérations :

- Interroger les données (SELECT) : récupérer des informations
- Modifier les données (INSERT, UPDATE, DELETE) : ajouter, modifier ou supprimer des données
- Définir la structure (CREATE, ALTER, DROP) : créer ou modifier des tables

Dans ce cours, nous nous concentrerons principalement sur l'interrogation des données avec SELECT, car c'est l'opération la plus fréquente pour un testeur débutant.

1.4 Avantages du SQL

- Langage standardisé : fonctionne sur la plupart des systèmes de bases de données (MySQL, Oracle, SQL Server, PostgreSQL, etc.)
- Syntaxe simple et lisible : proche du langage naturel en anglais
- Puissant : permet d'extraire des informations complexes rapidement
- Largement utilisé : compétence très demandée sur le marché du travail
- Gain de temps : permet d'automatiser la vérification des données

1.5 Présentation de la base de données SAKILA

Pour ce cours, nous utiliserons la base de données SAKILA, une base de démonstration fournie par MySQL.

Elle représente une entreprise de location de DVD.

Les principales tables que nous utiliserons :

- **actor** : liste des acteurs (prénom, nom)
- **film** : catalogue des films (titre, description, année, durée)
- **category** : catégories de films (Action, Comédie, etc.)
- **customer** : clients de la boutique
- **rental** : enregistrement des locations
- **payment** : paiements effectués
- **city** : villes
- **country** : pays

1.6 MySQL Workbench

MySQL Workbench est l'outil que nous utiliserons pour écrire et exécuter nos requêtes SQL.

C'est un logiciel gratuit qui offre :

- Un éditeur SQL avec coloration syntaxique
- La possibilité d'exécuter des requêtes et de visualiser les résultats
- Un explorateur de base de données pour voir la structure des tables

2. LES BASES DES REQUETES SQL

2.1 SELECT

Description : SELECT est la clause la plus importante en SQL. Elle permet de récupérer des données depuis une table.

Syntaxe	Exemple	Résultat attendu
SELECT colonne1, colonne2 **FROM** nom_table;	**SELECT** first_name, last_name **FROM** actor;	Cette requête affiche le prénom et le nom de tous les acteurs de la table **actor.**

Remarque : Pour sélectionner toutes les colonnes d'une table, utilisez l'astérisque (*) : **SELECT** * **FROM** actor ;

Bonne pratique : Sélectionnez uniquement les colonnes dont vous avez besoin pour optimiser les performances.

2.2 DISTINCT

Description : DISTINCT permet d'éliminer les doublons dans les résultats d'une requête.

Syntaxe	Exemple	Résultat attendu
SELECT DISTINCT colonne **FROM** nom_table ;	**SELECT DISTINCT** last_name **FROM** actor;	Cette requête affiche la liste des noms de famille des acteurs sans doublon. Si plusieurs acteurs portent le même nom, celui-ci n'apparaît qu'une seule fois.

Remarque : DISTINCT s'applique à l'ensemble des colonnes sélectionnées.

2.3 FROM

Description : FROM indique la table depuis laquelle on souhaite récupérer les données. C'est une clause obligatoire avec SELECT.

Syntaxe	Exemple	Résultat attendu
SELECT colonnes **FROM** nom_table ;	**SELECT** title **FROM** film ;	Cette requête affiche le titre de tous les films disponibles dans la table film.

Remarque : Le nom de la table doit être écrit exactement comme il apparaît dans la base de données.

2.4 WHERE

Description : WHERE permet de filtrer les résultats en spécifiant une condition. Seules les lignes qui respectent la condition seront retournées.

Syntaxe	Exemple	Résultat attendu
SELECT colonnes **FROM** nom_table **WHERE** condition;	**SELECT** first_name, last_name **FROM** actor **WHERE** first_name = 'PENELOPE' ;	Cette requête affiche uniquement les acteurs dont le prénom est PENELOPE.

Remarque : Les valeurs textuelles doivent être entourées de guillemets simples ('). SQL est sensible à la casse pour les données, mais pas toujours pour les mots-clés.

2.5 Opérateurs logiques : AND, OR, NOT

Description : Ces opérateurs permettent de combiner plusieurs conditions dans la clause WHERE.

AND

Syntaxe	Exemple	Résultat attendu
SELECT colonnes **FROM** nom_table **WHERE** condition1 **AND** condition2 ;	**SELECT** first_name, last_name **FROM** actor **WHERE** first_name = 'PENELOPE' **AND** last_name = 'GUINESS' ;	Affiche uniquement les acteurs dont le prénom est PENELOPE **ET** le nom GUINESS. Les deux conditions doivent être vraies.

OR

Syntaxe	Exemple	Résultat attendu
SELECT colonnes **FROM** nom_table **WHERE** condition1 **OR** condition2 ;	**SELECT** first_name, last_name **FROM** actor **WHERE** first_name = 'PENELOPE' OR first_name = 'NICK';	Affiche les acteurs dont le prénom est PENELOPE **OU** NICK. Au moins une des deux conditions doit être vraie.

NOT

Syntaxe	Exemple	Résultat attendu
SELECT colonnes **FROM** nom_table **WHERE** NOT condition;	**SELECT** first_name, last_name **FROM** actor **WHERE** NOT first_name = 'PENELOPE' ;	Affiche tous les acteurs sauf ceux qui s'appellent PENELOPE.

Bonne pratique : Utilisez des parenthèses pour clarifier l'ordre d'évaluation lorsque vous combinez AND et OR.

2.6 AS (alias)

Description : AS permet de renommer temporairement une colonne ou une table dans les résultats de la requête, pour une meilleure lisibilité.

Syntaxe	Exemple	Résultat attendu
SELECT colonne **AS** nouveau_nom **FROM** nom_table;	**SELECT** first_name **AS** prenom, last_name **AS** nom **FROM** actor;	Affiche les prénoms et noms des acteurs, mais les en-têtes de colonnes seront "prenom" et "nom" au lieu de "first_name" et "last_name".

Remarque : Le mot-clé AS est optionnel, mais il améliore la lisibilité. Vous pouvez écrire : SELECT first_name prenom FROM actor ;

2.7 COUNT

Description : COUNT est une fonction qui compte le nombre de lignes retournées par une requête.

Syntaxe	Exemple	Résultat attendu
SELECT COUNT(*) **FROM** nom_table ;	**SELECT** COUNT(*) **AS** nombre_acteurs **FROM** actor;	Affiche le nombre total d'acteurs dans la table actor.
SELECT COUNT(colonne) **FROM** nom_table ;	**SELECT** COUNT(*) **FROM** actor **WHERE** first_name = 'PENELOPE' ;	Compte le nombre d'acteurs qui s'appellent PENELOPE.

Remarque : COUNT(*) compte toutes les lignes, tandis que COUNT(colonne) ne compte que les lignes où la colonne n'est pas NULL.

2.8 LIMIT

Description : LIMIT permet de restreindre le nombre de lignes retournées par une requête. Très utile pour tester une requête ou afficher uniquement les premiers résultats.

Syntaxe	Exemple	Résultat attendu
SELECT colonnes **FROM** nom_table **LIMIT** nombre;	**SELECT** first_name, last_name **FROM** actor **LIMIT** 5;	Affiche uniquement les 5 lignes (les 5 premiers acteurs de la table.)

Remarque : Pratique pour visualiser un échantillon de données sans afficher l'ensemble de la table.

2.9 ORDER BY

Description : ORDER BY permet de trier les résultats selon une ou plusieurs colonnes. Par défaut, le tri est croissant (de A à Z, de 1 à 100).

Syntaxe	Exemple	Résultat attendu
SELECT colonnes **FROM** nom_table **ORDER BY** colonne;	**SELECT** first_name, last_name **FROM** actor **ORDER BY** last_name;	Affiche tous les acteurs triés par ordre alphabétique de nom de famille.

2.10 DESC

Description : DESC (descendant) est utilisé avec ORDER BY pour inverser l'ordre de tri (du plus grand au plus petit, de Z à A).

Syntaxe	Exemple	Résultat attendu
SELECT colonnes **FROM** nom_table **ORDER BY** colonne **DESC**;	**SELECT** first_name, last_name **FROM** actor **ORDER BY** last_name **DESC**;	Affiche les acteurs triés par nom de famille dans l'ordre décroissant (de Z à A).

Remarque : Pour un tri croissant explicite, vous pouvez utiliser ASC (ascendant), mais c'est le comportement par défaut.

2.11 LIKE

Description : LIKE permet de rechercher des données en utilisant des motifs (patterns). Très utile pour les recherches partielles.

Caractères spéciaux :

- % : représente zéro, un ou plusieurs caractères
- _ : représente exactement un caractère

Syntaxe	Exemple	Résultat attendu
SELECT colonnes **FROM** nom_table **WHERE** colonne LIKE 'motif' ;	**SELECT** first_name, last_name **FROM** actor **WHERE** first_name LIKE 'P%' ;	Affiche tous les acteurs dont le prénom commence par la lettre P.
	SELECT first_name, last_name **FROM** actor **WHERE** first_name LIKE %A' ;	*Prénoms se terminant par 'A'*
	SELECT first_name **FROM** actor **WHERE** first_name LIKE '%EN%' ;	*Prénoms contenants 'EN'*
	SELECT first_name **FROM** actor **WHERE** first_name LIKE 'N___';	*Prénoms de 4 lettres commençant par 'N'*

Bonne pratique : LIKE peut être lent sur de grandes tables. Utilisez-le uniquement quand nécessaire.

2.12 IN

Description : IN permet de spécifier plusieurs valeurs possibles dans une condition WHERE. C'est un raccourci pour éviter d'utiliser plusieurs OR.

Syntaxe	Exemple	Résultat attendu
SELECT colonnes **FROM** nom_table **WHERE** colonne IN (valeur1, valeur2, valeur3);	**SELECT** first_name, last_name **FROM** actor **WHERE** first_name IN ('PENELOPE', 'NICK', 'ED');	Affiche tous les acteurs dont le prénom est PENELOPE, NICK ou ED.

Equivalent avec OR :

```
SELECT first_name, last_name FROM actor
WHERE first_name = 'PENELOPE' OR first_name = 'NICK' OR first_name = 'ED';
```

Remarque : IN rend le code plus lisible et plus court quand on a beaucoup de valeurs.

2.13 NOT IN

Description : NOT IN est l'inverse de IN. Il permet d'exclure plusieurs valeurs.

Syntaxe	Exemple	Résultat attendu
SELECT colonnes **FROM** nom_table **WHERE** colonne NOT IN (valeur1, valeur2);	**SELECT** first_name, last_name **FROM** actor **WHERE** first_name NOT IN ('PENELOPE', 'NICK');	Affiche tous les acteurs sauf ceux dont le prénom est PENELOPE ou NICK.

2.14 BETWEEN

Description : BETWEEN permet de sélectionner des valeurs dans un intervalle donné (nombres, dates, texte). Les bornes sont inclusives.

Syntaxe	Exemple	Résultat attendu
SELECT colonnes **FROM** nom_table **WHERE** colonne BETWEEN valeur1 AND valeur2;	**SELECT** title, length **FROM** film **WHERE** length BETWEEN 100 AND 120;	Affiche les films dont la durée est comprise entre 100 et 120 minutes (inclus).

Equivalent avec des opérateurs :

SELECT title, length **FROM** film **WHERE** length >= 100 **AND** length <= 120;

Remarque : BETWEEN fonctionne également avec des dates et des chaînes de caractères.

2.15 NOT BETWEEN

Description : NOT BETWEEN exclut les valeurs comprises dans un intervalle.

Syntaxe	Exemple	Résultat attendu
SELECT colonnes **FROM** nom_table **WHERE** colonne NOT BETWEEN valeur1 AND valeur2;	**SELECT** title, length **FROM** film **WHERE** length NOT BETWEEN 100 AND 120;	Affiche les films dont la durée est inférieure à 100 minutes ou supérieure à 120 minutes.

3. FONCTIONS SQL

Les fonctions SQL permettent d'effectuer des calculs ou des transformations sur les données.

3.1 Fonctions d'agrégation

Les fonctions d'agrégation effectuent un calcul sur un ensemble de valeurs et retournent une valeur unique.

AVG

Description : AVG calcule la moyenne d'une colonne numérique.

Syntaxe	Exemple	Résultat attendu
SELECT AVG(colonne) **FROM** nom_table;	**SELECT** AVG(length) **AS** duree_moyenne **FROM** film;	Affiche la durée moyenne de tous les films en minutes.

Remarque : AVG ignore les valeurs NULL dans le calcul.

MIN

Description : MIN retourne la valeur minimale d'une colonne.

Syntaxe	Exemple	Résultat attendu
SELECT MIN(colonne) **FROM** nom_table;	**SELECT** MIN(length) **AS** film_le_plus_court **FROM** film;	Affiche la durée du film le plus court de la base de données.

Remarque : MIN fonctionne avec des nombres, des dates et même du texte (ordre alphabétique).

MAX

Description : MAX retourne la valeur maximale d'une colonne.

Syntaxe	Exemple	Résultat attendu
SELECT MAX(colonne) **FROM** nom_table;	**SELECT** MAX(length) **AS** film_le_plus_long **FROM** film;	Affiche la durée du film le plus long de la base de données.

Remarque : MAX fonctionne avec des nombres, des dates et même du texte (ordre alphabétique).

SUM

Description : SUM calcule la somme des valeurs d'une colonne numérique.

Syntaxe	Exemple	Résultat attendu
SELECT SUM(colonne) **FROM** nom_table;	**SELECT** SUM(amount) **AS** total_paiements **FROM** payment;	Affiche le montant total de tous les paiements effectués.

Remarque : Très utile pour calculer des totaux (chiffre d'affaires, quantités, etc.).

3.2 Fonctions scalaires

Les fonctions scalaires opèrent sur une valeur et retournent une valeur pour chaque ligne.

FLOOR

Description : FLOOR arrondit un nombre à l'entier inférieur (vers le bas).

Syntaxe	Exemple	Résultat attendu
SELECT FLOOR(nombre);	**SELECT** title, length, FLOOR(length / 60) **AS** duree_heures **FROM** film;	Affiche le titre et la durée de chaque film en minutes, ainsi que la durée en heures (arrondie à l'entier inférieur).

Exemple de calcul : Si length = 125 minutes, FLOOR(125/60) = FLOOR(2.08) = 2 heures.

ROUND

Description : ROUND arrondit un nombre au nombre de décimales spécifié.

Syntaxe	Exemple	Résultat attendu
SELECT ROUND(nombre, decimales);	**SELECT** title, rental_rate, ROUND(rental_rate * 1.20, 2) **AS** prix_avec_taxe **FROM** film;	Affiche le titre, le prix de location et le prix avec 20% de taxes, arrondi à 2 décimales.

Remarque : Si vous omettez le deuxième paramètre, ROUND arrondit à l'entier le plus proche.

LENGTH

Description : LENGTH retourne le nombre de caractères d'une chaîne de texte.

Syntaxe	Exemple	Résultat attendu
SELECT LENGTH(colonne) **FROM** nom_table;	**SELECT** title, LENGTH(title) **AS** longueur_titre **FROM** film;	Affiche le titre de chaque film et le nombre de caractères qu'il contient.

Exemple : Le titre "ACADEMY DINOSAUR" contient 16 caractères (espaces compris).

CONCAT

Description : CONCAT permet de concaténer (assembler) plusieurs chaînes de caractères en une seule.

Syntaxe	Exemple	Résultat attendu
SELECT CONCAT(chaine1, chaine2, ...) **FROM** nom_table;	**SELECT** CONCAT(first_name, ' ', last_name) **AS** nom_complet **FROM** actor;	Affiche le nom complet de chaque acteur en assemblant prénom, espace et nom dans une seule colonne.

Exemple de résultat : "PENELOPE GUINESS", "NICK WAHLBERG", etc.

Bonne pratique : Très utile pour formater l'affichage de données.

4. REGROUPEMENT DES DONNEES

4.1 GROUP BY

Description : GROUP BY permet de regrouper des lignes qui ont des valeurs identiques dans certaines colonnes. Il est généralement utilisé avec des fonctions d'agrégation.

Syntaxe	Exemple	Résultat attendu
SELECT colonne, fonction_agregation(colonne2) **FROM** nom_table **GROUP BY** colonne;	**SELECT** first_name, COUNT(*) **AS** nombre **FROM** actor **GROUP BY** first_name;	Affiche chaque prénom unique et le nombre d'acteurs portant ce prénom. Exemple de résultat : • PENELOPE : 2 • NICK : 3 • ED : 1

Remarque : Toutes les colonnes du SELECT (sauf celles dans les fonctions d'agrégation) doivent apparaître dans le GROUP BY.

4.2 HAVING

Description : HAVING permet de filtrer les groupes créés par GROUP BY. C'est l'équivalent du WHERE, mais pour les groupes.

Syntaxe	Exemple	Résultat attendu
SELECT colonne, fonction_agregation(colonne2) **FROM** nom_table **GROUP BY** colonne **HAVING** condition_sur_agregation;	**SELECT** first_name, COUNT(*) **AS** nombre **FROM** actor **GROUP BY** first_name **HAVING** COUNT(*) > 2;	Affiche uniquement les prénoms portés par plus de 2 acteurs.

4.3 Différence entre WHERE et HAVING

WHERE :	HAVING :
• Filtre les lignes AVANT le regroupement • Ne peut pas utiliser de fonctions d'agrégation • S'applique sur les colonnes de la table *Avec WHERE : on filtre d'abord, puis on regroupe*	• Filtre les groupes APRÈS le regroupement • Utilise des fonctions d'agrégation • S'applique sur les résultats de GROUP BY *-- Avec HAVING : on regroupe d'abord, puis on filtre les groupes*
Exemple comparatif : **SELECT** first_name, COUNT(*) **AS** nombre **FROM** actor **WHERE** last_name LIKE 'A%' **GROUP BY** first_name;	**SELECT** first_name, COUNT(*) **AS** nombre **FROM** actor **GROUP BY** first_name **HAVING** COUNT(*) >= 3;
Résultat : Compte les prénoms, mais uniquement pour les acteurs dont le nom commence par A.	Résultat : Affiche uniquement les prénoms portés par au moins 3 acteurs.

Bonne pratique : Utilisez WHERE pour filtrer les données avant regroupement (plus performant) et HAVING pour filtrer sur les résultats d'agrégation.

5. LES JOINTURES

5.1 Présentation du concept de jointure

Dans une base de données relationnelle, les informations sont réparties dans plusieurs tables pour éviter la redondance. Les jointures permettent de combiner les données de plusieurs tables en une seule requête.

Exemple :

- La table city contient les villes
- La table country contient les pays
- Pour afficher une ville avec son pays, on doit faire une jointure

Chaque ville dans la table city possède un identifiant de pays (country_id) qui fait référence à la table country. C'est ce qu'on appelle une clé étrangère.

5.2 INNER JOIN

Description : INNER JOIN retourne uniquement les lignes pour lesquelles il existe une correspondance dans les deux tables.

Syntaxe :

Syntaxe	Exemple	Résultat attendu
SELECT colonnes **FROM** table1 **INNER JOIN** table2 **ON** table1.colonne = table2.colonne;	**SELECT** city.city, country.country **FROM** city **INNER JOIN** country **ON** city.country_id = country.country_id **LIMIT** 10;	Affiche les 10 premières villes avec leur pays correspondant. Exemple de résultat : • A Corua (La Corua) : Spain • Abha : Saudi Arabia • Abu Dhabi : United Arab Emirates • ...

Remarque : Si une ville n'a pas de pays correspondant (ou inversement), elle n'apparaîtra pas dans le résultat.

5.3 ON

Description : ON spécifie la condition de jointure, c'est-à-dire comment les deux tables sont liées entre elles.

Syntaxe :

Syntaxe	Exemple	Résultat attendu
... **INNER JOIN** table2 **ON** table1.colonne_cle = table2.colonne_cle	**SELECT** customer.first_name, customer.last_name, city.city **FROM** customer **INNER JOIN** address **ON** customer.address_id = address.address_id **INNER JOIN** city **ON** address.city_id = city.city_id **LIMIT** 5;	Affiche les 5 premiers clients avec leur ville. Cette requête utilise deux jointures successives pour relier customer -> address -> city.

Bonne pratique : Précisez toujours le nom de la table avant le nom de la colonne (table.colonne) pour éviter les ambiguïtés quand plusieurs tables ont des colonnes de même nom.

5.4 Exemple complet de jointure

Objectif : Afficher le titre du film et sa catégorie.

Contexte	Requête	Résultat attendu
• La table film contient les films • La table category contient les catégories • La table film_category fait le lien entre les deux (table de liaison)	**SELECT** film.title, category.name **AS** categorie **FROM** film **INNER JOIN** film_category **ON** film.film_id = film_category.film_id **INNER JOIN** category **ON** film_category.category_id = category.category_id **LIMIT** 10;	Affiche les 10 premiers films avec leur catégorie. Exemple de résultat : • ACADEMY DINOSAUR : Documentary • ACE GOLDFINGER : Horror • ADAPTATION HOLES : Documentary

Explication :

1. On part de la table film
2. On la relie à film_category pour savoir quelle catégorie appartient à chaque film
3. On relie film_category à category pour obtenir le nom de la catégorie

6. LES SOUS-REQUETES

6.1 Qu'est-ce qu'une sous-requête

Une sous-requête (ou requête imbriquée) est une requête SQL écrite à l'intérieur d'une autre requête. Elle permet de décomposer un problème complexe en plusieurs étapes.

La sous-requête est exécutée en premier, puis son résultat est utilisé par la requête principale.

Structure :

```
SELECT colonnes
FROM table
WHERE colonne OPERATEUR (SELECT colonne FROM table WHERE condition) ;
```

6.2 Exemple simple de sous-requête

Objectif	Requête	Résultat attendu
Trouver tous les films dont la durée est supérieure à la durée moyenne.	**SELECT** title, length **FROM** film **WHERE** length > (**SELECT** AVG(length) **FROM** film);	Affiche tous les films qui durent plus longtemps que la moyenne

Explication :

1. La sous-requête (SELECT AVG(length) FROM film) calcule la durée moyenne de tous les films (environ 115 minutes)
2. La requête principale compare chaque film à cette moyenne
3. Seuls les films dont la durée dépasse la moyenne sont affichés

Remarque : La sous-requête doit retourner une seule valeur quand elle est utilisée avec des opérateurs de comparaison (>, <, =).

6.3 Quand utiliser une sous-requête

Les sous-requêtes sont utiles quand :

- Vous devez comparer une valeur à un résultat calculé (comme une moyenne)
- Vous devez filtrer sur des données provenant d'une autre table
- Votre requête est plus lisible décomposée en plusieurs étapes

Bonne pratique : Pour des requêtes simples, préférez les jointures qui sont généralement plus performantes. Les sous-requêtes sont plus adaptées pour des calculs ou des filtres complexes.

RESUME FINAL

Ce que vous avez appris

Vous maîtrisez maintenant les fondamentaux du SQL :

1. Interroger des données avec SELECT, FROM, WHERE
2. Filtrer avec DISTINCT, LIKE, IN, BETWEEN
3. Trier et limiter avec ORDER BY, LIMIT
4. Compter et calculer avec COUNT, AVG, MIN, MAX, SUM
5. Transformer avec FLOOR, ROUND, LENGTH, CONCAT
6. Regrouper avec GROUP BY et filtrer les groupes avec HAVING
7. Combiner des tables avec INNER JOIN
8. Utiliser des sous-requêtes pour des filtres avancés

Pour aller plus loin

Pour progresser, pratiquez régulièrement sur la base SAKILA :

- Essayez de répondre à des questions métier avec SQL
- Combinez plusieurs clauses dans une même requête
- Explorez les autres tables de SAKILA

Conseils pour les testeurs

En tant que testeur, utilisez le SQL pour :

- Vérifier rapidement si une donnée existe dans la base
- Compter les enregistrements avant et après une action
- Identifier des incohérences de données
- Préparer des jeux de tests variés
- Gagner du temps dans vos vérifications

Le SQL est un outil puissant qui rendra votre travail de testeur plus efficace et plus autonome.

FIN DU COURS