Cours d'introduction à SQL

Abdelwakil Benabdi - Prépa+

10 juillet 2025

1 Introduction

SQL (Structured Query Language) est un langage standard utilisé pour interagir avec les bases de données relationnelles. Il permet de créer, modifier et interroger des bases de données. Les bases de données sont omniprésentes dans l'informatique moderne : elles sont utilisées dans les sites web, les applications mobiles, les systèmes d'information d'entreprise, etc.

Ce cours a pour objectif de vous initier à la création de bases de données en SQL, et plus précisément à la création de **tables**, à la **définition des relations** entre elles (clés primaires et étrangères), et à la gestion de **données avec contraintes**.

Nous aborderons également les différentes commandes SQL fondamentales telles que `CREATE`, `INSERT`, `SELECT`, `UPDATE` et `DELETE`, qui permettent de manipuler les données efficacement dans une base.

2 Création d'une table SQL

Une table en SQL est une structure qui permet de stocker des données sous forme de lignes (enregistrements) et de colonnes (champs). Chaque table représente une entité (par exemple : un étudiant, un cours, une inscription).

Voici la structure de base pour créer une table en SQL :

```
CREATE TABLE nom_table (
    nom_colonne1 TYPE1 CONTRAINTES,
    nom_colonne2 TYPE2 CONTRAINTES,
    ...
);
```

Chaque colonne doit avoir :

— un nom,
— un type (par exemple : INTEGER, TEXT, DATE, REAL),
— et éventuellement des contraintes (NOT NULL, PRIMARY KEY, UNIQUE, CHECK, etc.).

3 Création des tables dans SQLite

Avant de créer les tables, il est recommandé d'activer la gestion des clés étrangères dans SQLite :

```
PRAGMA foreign_keys = ON;
```

Ensuite, on peut créer les trois tables suivantes : **etudiants**, **cours** et **inscriptions**.

Table des étudiants

```
CREATE TABLE etudiants (
    id_etudiant INTEGER PRIMARY KEY AUTOINCREMENT,
    nom TEXT NOT NULL,
    prenom TEXT NOT NULL
);
```

Table des cours

```
CREATE TABLE cours (
    id_cours INTEGER PRIMARY KEY AUTOINCREMENT,
    nom_cours TEXT NOT NULL,
    enseignant TEXT NOT NULL
);
```

Table des inscriptions

Cette table relie les étudiants aux cours et contient également leur note.

```
CREATE TABLE inscriptions (
    id_inscription INTEGER PRIMARY KEY AUTOINCREMENT,
    id_etudiant INTEGER,
    id_cours INTEGER,
    date_inscription DATE,
    note REAL CHECK(note BETWEEN 0 AND 20),
    FOREIGN KEY(id_etudiant) REFERENCES etudiants(id_etudiant),
    FOREIGN KEY(id_cours) REFERENCES cours(id_cours)
);
```

4 Insertion de données

La commande `INSERT INTO` permet d'ajouter une ou plusieurs lignes dans une table. Sa syntaxe de base est la suivante :

```
INSERT INTO nom_table (colonne1, colonne2, ...)
VALUES (valeur1, valeur2, ...);
```

- `nom_table` : le nom de la table dans laquelle on insère les données.
- `(colonne1, colonne2, ...)` : la liste des colonnes à remplir.
- `(valeur1, valeur2, ...)` : les valeurs à insérer, dans le même ordre que les colonnes.

Étudiants

```
INSERT INTO etudiants (nom, prenom) VALUES ('Durand', 'Alice');
INSERT INTO etudiants (nom, prenom) VALUES ('Martin', 'Bob');
INSERT INTO etudiants (nom, prenom) VALUES ('Lemoine', 'Chlo');
INSERT INTO etudiants (nom, prenom) VALUES ('Nguyen', 'David');
```

Cours

```
INSERT INTO cours (nom_cours, enseignant) VALUES ('Mathmatiques', 'Mr. Dupuis');
INSERT INTO cours (nom_cours, enseignant) VALUES ('Physique', 'Mme. Petit');
INSERT INTO cours (nom_cours, enseignant) VALUES ('Chimie', 'Dr. Laurent');
INSERT INTO cours (nom_cours, enseignant) VALUES ('Informatique', 'Mme. Lemoine');
INSERT INTO cours (nom_cours, enseignant) VALUES ('Franais', 'Mr. Moreau');
```

Inscriptions

```
INSERT INTO etudiants (nom, prenom) VALUES ('Durand', 'Alice');
INSERT INTO etudiants (nom, prenom) VALUES ('Martin', 'Bob');
INSERT INTO etudiants (nom, prenom) VALUES ('Lemoine', 'Chloe');
INSERT INTO etudiants (nom, prenom) VALUES ('Nguyen', 'David');
```

Cours

```
INSERT INTO cours (nom_cours, enseignant) VALUES ('Mathematiques', 'Mr. Dupuis');
INSERT INTO cours (nom_cours, enseignant) VALUES ('Physique', 'Mme. Petit');
INSERT INTO cours (nom_cours, enseignant) VALUES ('Chimie', 'Dr. Laurent');
INSERT INTO cours (nom_cours, enseignant) VALUES ('Informatique', 'Mme. Lemoine');
INSERT INTO cours (nom_cours, enseignant) VALUES ('Francais', 'Mr. Moreau');
```

Inscriptions

```
-- Alice (id_etudiant = 1)
INSERT INTO inscriptions (id_etudiant, id_cours, date_inscription, note)
VALUES (1, 1, '2025-09-01', 17.5);
INSERT INTO inscriptions (id_etudiant, id_cours, date_inscription, note)
VALUES (1, 2, '2025-09-01', 14.0);
INSERT INTO inscriptions (id_etudiant, id_cours, date_inscription, note)
VALUES (1, 4, '2025-09-01', 16.5);

-- Bob (id_etudiant = 2)
INSERT INTO inscriptions (id_etudiant, id_cours, date_inscription, note)
VALUES (2, 1, '2025-09-02', 12.5);
INSERT INTO inscriptions (id_etudiant, id_cours, date_inscription, note)
VALUES (2, 3, '2025-09-02', 15.0);
INSERT INTO inscriptions (id_etudiant, id_cours, date_inscription, note)
VALUES (2, 5, '2025-09-02', 13.0);

-- Chloe (id_etudiant = 3)
INSERT INTO inscriptions (id_etudiant, id_cours, date_inscription, note)
VALUES (3, 2, '2025-09-03', 18.0);
INSERT INTO inscriptions (id_etudiant, id_cours, date_inscription, note)
VALUES (3, 3, '2025-09-03', 19.0);
INSERT INTO inscriptions (id_etudiant, id_cours, date_inscription, note)
VALUES (3, 4, '2025-09-03', 17.0);

-- David (id_etudiant = 4)
INSERT INTO inscriptions (id_etudiant, id_cours, date_inscription, note)
VALUES (4, 1, '2025-09-04', 11.5);
INSERT INTO inscriptions (id_etudiant, id_cours, date_inscription, note)
VALUES (4, 2, '2025-09-04', 12.0);
```

```sql
INSERT INTO inscriptions (id_etudiant, id_cours, date_inscription, note)
VALUES (4, 5, '2025-09-04', 10.0);
```

5 Exercices : extraction de données

Dans les exercices suivants, écrivez la requête SQL permettant d'obtenir les résultats demandés à partir des tables créées précédemment. Vous pouvez utiliser les mots-clés `SELECT`, `FROM`, `WHERE`, `LIMIT`, etc.

Exercice 1 : tous les prénoms des étudiants

Afficher uniquement les prénoms de tous les étudiants.

Réponse :

Exercice 2 : cours donnés par Mme. Lemoine

Afficher les noms des cours assurés par `Mme. Lemoine`.

Réponse :

Exercice 3 : notes supérieures ou égales à 15

Afficher les prénoms des étudiants ayant obtenu une note supérieure ou égale à 15, ainsi que leur note.

Réponse :

Exercice 4 : cours suivis par l'étudiant numéro 2

Afficher les noms des cours auxquels est inscrit l'étudiant ayant l'identifiant `id_etudiant = 2`.

Réponse :

Exercice 5 : 3 premières lignes de la table inscriptions

Afficher toutes les colonnes pour les 3 premières lignes de la table `inscriptions`.

Réponse :

6 Suppression de données

Il est possible en SQL de supprimer soit certaines lignes d'une table, soit la table entière.

Suppression de lignes

Pour supprimer une ou plusieurs lignes d'une table, on utilise la commande `DELETE`. La syntaxe générale est :

```sql
DELETE FROM nom_table
WHERE condition;
```

Par exemple, pour supprimer toutes les lignes de la table `inscriptions` dont la date est antérieure au 14 juin 2022 :

```sql
DELETE FROM inscriptions
WHERE date_inscription < '2022-06-14';
```

Remarque : les dates sont traitées comme des chaînes de caractères. Il est donc essentiel de respecter un format de date cohérent (comme `aaaa-mm-jj`) pour que les comparaisons fonctionnent correctement.

Suppression de toute une table

Pour supprimer complètement une table (et toutes les données qu'elle contient), on utilise la commande `DROP TABLE` :

```
DROP TABLE inscriptions;
```

Cette commande efface la structure et le contenu de la table de façon définitive. Elle est à utiliser avec précaution.

7 Modification de données

Il est possible de modifier une ou plusieurs lignes dans une table à l'aide de la commande `UPDATE`. Cette instruction permet de mettre à jour les valeurs d'une ou plusieurs colonnes en fonction d'une condition donnée.

Syntaxe générale

```
UPDATE nom_table
SET colonne = nouvelle_valeur
WHERE condition;
```

Exemple 1 : modifier le prénom d'un étudiant

Imaginons que l'on souhaite corriger une faute de frappe dans le prénom d'un étudiant :

```
UPDATE etudiants
SET prenom = 'Chloe'
WHERE id_etudiant = 3;
```

Cette requête met à jour le prénom de l'étudiant ayant l'identifiant 3.

Exemple 2 : corriger une note

Supposons que la note d'un étudiant ait été mal enregistrée. On peut la corriger ainsi :

```
UPDATE inscriptions
SET note = 18.5
WHERE id_etudiant = 1 AND id_cours = 2;
```

Ici, on modifie la note de l'étudiante numéro 1 pour le cours numéro 2.

Attention : l'oubli de la clause `WHERE` entraînerait la modification de **toutes les lignes** de la table, ce qui est généralement une erreur.

Vérification

Pour vérifier que la modification a bien été faite, on peut exécuter une requête `SELECT` ciblée :

```
SELECT * FROM inscriptions
WHERE id_etudiant = 1 AND id_cours = 2;
```

8 Fonctions d'agrégation

Il est possible de faire des opérations d'agrégation sur les colonnes à l'aide de la commande `SELECT`. Les fonctions d'agrégation les plus courantes sont :

- `COUNT` : compter le nombre de lignes,
- `SUM` : faire la somme d'une colonne numérique,
- `AVG` : calculer la moyenne,
- `MIN` : trouver la plus petite valeur,
- `MAX` : trouver la plus grande valeur.

Exemple 1 : moyenne des notes

```
SELECT AVG(note) FROM inscriptions;
```

Cette requête retourne la moyenne générale des notes de tous les étudiants, tous cours confondus.

Exemple 2 : nombre d'étudiants inscrits à un cours

```
SELECT COUNT(*) FROM inscriptions
WHERE id_cours = 1;
```

Cela permet de connaître le nombre d'étudiants inscrits au cours numéro 1.

Exemple 3 : note maximale obtenue dans le cours 2

```
SELECT MAX(note) FROM inscriptions
WHERE id_cours = 2;
```

Exercices – Fonctions d'agrégation

Exercice 1 : Afficher le nombre total d'inscriptions dans la base.

Réponse :

Exercice 2 : Afficher la note minimale obtenue par un étudiant.

Réponse :

Exercice 3 : Afficher la moyenne des notes pour l'étudiant ayant l'`id_etudiant = 2`.

Réponse :

Exercice 4 : Compter combien de cours existent dans la table `cours`.

Réponse :

Exercice 5 : Afficher la plus grande note obtenue dans le cours d'informatique (id = 4).

Réponse :

9 Les jointures (JOIN)

En SQL, une jointure permet de croiser des données provenant de plusieurs tables en liant des colonnes communes, souvent des clés étrangères. L'instruction `JOIN` (ou `INNER JOIN`) est la plus courante : elle retourne uniquement les lignes présentes dans les deux tables liées.

Syntaxe générale

```
SELECT colonnes
FROM table1
JOIN table2
ON table1.cle = table2.cle;
```

Exemple : afficher les prénoms des étudiants avec les noms des cours suivis

```
SELECT e.prenom, c.nom_cours
FROM etudiants e
JOIN inscriptions i ON e.id_etudiant = i.id_etudiant
JOIN cours c ON i.id_cours = c.id_cours;
```

Cette requête affiche tous les prénoms d'étudiants associés aux noms des cours auxquels ils sont inscrits.

Exercices – Jointures

Exercice 1 : Afficher les noms et prénoms des étudiants ainsi que la note qu'ils ont obtenue pour chaque cours.

Réponse :

Exercice 2 : Afficher les noms des cours et les enseignants pour lesquels au moins un étudiant est inscrit.

Réponse :

Exercice 3 : Afficher les prénoms des étudiants et la date d'inscription au cours d'informatique (id = 4).

Réponse :

Exercice 4 : Afficher la liste des cours (nom) avec le nombre d'étudiants inscrits à chacun.

Réponse :

Exercice 5 : Afficher, pour chaque étudiant, son prénom et la moyenne de ses notes.

Réponse :

10 Opérations ensemblistes : UNION, INTERSECT et EXCEPT

SQL permet de combiner les résultats de plusieurs requêtes grâce à des opérations ensemblistes. Ces opérations s'appliquent sur des ensembles de lignes provenant de requêtes `SELECT`.

- `UNION` : fusionne les résultats de deux requêtes sans doublons.
- `INTERSECT` : retourne les lignes communes aux deux requêtes.
- `EXCEPT` : retourne les lignes présentes dans la première requête mais absentes dans la seconde.

Remarques importantes :

- Les requêtes combinées doivent avoir le même nombre de colonnes.
- Les colonnes doivent avoir des types compatibles (par exemple : deux colonnes de type texte ou deux colonnes numériques).
- Les noms de colonnes de la première requête sont utilisés pour nommer les résultats.

Exemple :

Afficher tous les prénoms d'étudiants et tous les enseignants (sans doublons) dans une seule colonne.

```
SELECT prenom FROM etudiants
UNION
SELECT enseignant FROM cours;
```

Remarque : les valeurs communes (mêmes prénoms/enseignants) n'apparaîtront qu'une seule fois.

Exercices – UNION, INTERSECT, EXCEPT

Exercice 1 : Afficher tous les prénoms des étudiants et tous les noms des enseignants dans une seule liste, sans doublons.

Réponse :

Exercice 2 : Afficher les noms des cours qui sont suivis par au moins un étudiant ET qui sont enseignés par `Mme. Petit`.

Réponse :

Exercice 3 : Afficher les prénoms des étudiants qui ne sont inscrits à aucun cours.

Réponse :

Exercice 4 : Afficher les prénoms des étudiants qui sont inscrits à des cours, mais pas au cours d'informatique (`id_cours = 4`).

Réponse :