SQL : Gestion et Requêtes de Bases de Données

Prérequis

Ce cours nécessite une connaissance de base en informatique, notamment la manipulation de fichiers et de dossiers. Il s'inscrit dans la continuité des chapitres sur la programmation et la structure des données en NSI, en terminale. La compréhension des concepts de tableaux et de structures de données est fortement recommandée. Ce cours prépare à la conception et à la manipulation de bases de données relationnelles.

Chapitre 1 : Introduction à SQL et aux Bases de Données Relationnelles

1.1 Définition d'une Base de Données

Une **base de données** est un ensemble organisé d'informations structurées et stockées électroniquement. Elle permet de stocker, gérer et récupérer des données de manière efficace. Une **base de données relationnelle** est un type de base de données qui organise les données dans des tables avec des lignes (enregistrements) et des colonnes (champs). Chaque colonne possède un type de données (entier, texte, date, etc.).

1.2 Le Langage SQL

SQL (Structured Query Language) est un langage standard utilisé pour interagir avec les bases de données relationnelles. Il permet de créer, modifier, interroger et administrer les bases de données. Il est composé de plusieurs instructions (requêtes) permettant de réaliser des opérations spécifiques. On utilisera ici un système de gestion de bases de données (SGBD) tel que MySQL, PostgreSQL ou SQLite.

1.3 Création d'une Table

Pour créer une table dans une base de données, on utilise l'instruction `CREATE TABLE`. L'exemple suivant illustre la création d'une table nommée “étudiants” avec les champs “id” (entier), “nom” (texte) et “note” (entier) :

```sql
CREATE TABLE étudiants (
 id INT PRIMARY KEY,
 nom VARCHAR(255),
```

```
 note INT
);
```

`PRIMARY KEY` indique que le champ “id” est la clé primaire, unique pour chaque enregistrement. `VARCHAR(255)` spécifie que le champ “nom” peut contenir une chaîne de caractères de longueur maximum 255.

Exercice 1 : Créez une table nommée “livres” avec les champs “titre” (texte), “auteur” (texte) et “année” (entier). Indiquez une clé primaire appropriée.

Corrigé guidé Exercice 1 : Il faut choisir un champ comme clé primaire. On peut utiliser un champ `id` entier auto-incrémenté (pour simplifier), ou bien on peut supposer que le titre et l'auteur forment une clé primaire (si on est sûr qu'un livre n'aura jamais deux éditions distinctes avec le même titre et le même auteur). Voici une solution possible avec un `id` auto-incrémenté:

```sql
CREATE TABLE livres (
 id INT PRIMARY KEY AUTO_INCREMENT,
 titre VARCHAR(255),
 auteur VARCHAR(255),
 année INT
);
```

Chapitre 2 : Requêtes SQL de sélection et de filtrage

2.1 Sélection de données avec `SELECT`

L'instruction `SELECT` permet de récupérer des données d'une ou plusieurs tables. Par exemple, pour sélectionner tous les champs de la table “étudiants”, on utilise :

```sql
SELECT * FROM étudiants;
```

Pour sélectionner uniquement certains champs, on les liste après `SELECT` :

```sql
SELECT id, nom FROM étudiants;
```

2.2 Filtrage de données avec `WHERE`

La clause `WHERE` permet de filtrer les résultats en fonction d'une condition. Par exemple, pour sélectionner les étudiants ayant une note supérieure à 15 :

```sql
SELECT * FROM étudiants WHERE note > 15;
```

On peut combiner plusieurs conditions avec les opérateurs `AND` et `OR`.

Exercice 2 : Sélectionnez les livres publiés après l'année 2000 de la table “livres” créée à l'exercice 1.

Corrigé guidé Exercice 2 :

```sql
SELECT * FROM livres WHERE année > 2000;
```

Chapitre 3 : Requêtes SQL avancées (Agrégation et Jointures)

3.1 Agrégation de données avec `COUNT`, `AVG`, `SUM`, `MAX`, `MIN`

Les fonctions d'agrégation permettent de calculer des valeurs à partir d'un ensemble de données. `COUNT` compte le nombre d'enregistrements, `AVG` calcule la moyenne, `SUM` la somme, `MAX` la valeur maximale et `MIN` la valeur minimale. Exemple : compter le nombre d'étudiants:

```sql
SELECT COUNT(*) FROM étudiants;
```

3.2 Jointures de tables

Les jointures permettent de combiner des données de plusieurs tables. La jointure `INNER JOIN` retourne uniquement les lignes ayant une correspondance dans les deux tables.

Imaginons une table “cours” avec les champs “id_cours”, “nom_cours” et “id_etudiant”. Une jointure permet de combiner les données des tables “étudiants” et “cours”.

```sql
SELECT étudiants.nom, cours.nom_cours
```

```
FROM étudiants
INNER JOIN cours ON étudiants.id = cours.id_etudiant;
```

Cette requête affiche le nom de chaque étudiant et le nom du cours qu'il suit.

Résumé

- **Base de données:** Ensemble organisé d'informations structurées.
- **Base de données relationnelle:** Organise les données en tables avec lignes et colonnes.
- **SQL (Structured Query Language):** Langage standard pour interagir avec les bases de données relationnelles.
- **`CREATE TABLE`:** Instruction pour créer une table.
- **`PRIMARY KEY`:** Champ unique identifiant chaque enregistrement.
- **`SELECT`:** Instruction pour récupérer des données.
- **`WHERE`:** Clause pour filtrer les résultats.
- **`COUNT(*)`:** Fonction pour compter le nombre d'enregistrements.
- **`AVG()`:** Fonction pour calculer la moyenne.
- **`INNER JOIN`:** Jointure pour combiner des données de plusieurs tables.
- Le chapitre 1 introduit les concepts de base de données et de SQL, ainsi que la création de tables.
- Le chapitre 2 explique les requêtes de sélection et de filtrage avec `SELECT` et `WHERE`.
- Le chapitre 3 présente les requêtes avancées avec les fonctions d'agrégation et les jointures.