Introduction à SQL et requêtes de bases

Introduction à SQL

Qu'est-ce que SQL?

SQL (Structured Query Language) est un langage de programmation utilisé pour gérer et manipuler des **bases de données relationnelles**.

Avantages de SQL	Exemples d'utilisation
Facilité d'apprentissage	- Recherche d'informations
Flexibilité	- Ajout de nouvelles données
Adaptabilité	- Mise à jour de données
Conformité aux normes	- Suppression de données

Introduction à SQL et requêtes de base

Requêtes de base

CREATE

La commande `CREATE` en SQL est essentielle pour créer des structures de données dans une base de données. La création d'une table est l'une des utilisations les plus courantes de cette commande. Voici la syntaxe générale de la commande `CREATE TABLE` :

```
CREATE TABLE table_name
(
    column1 datatype,
    column2 datatype,
    column3 datatype,
    ...
);
```

- `CREATE TABLE` indique que nous allons créer une nouvelle table.
- `table_name` est le nom de la table que vous souhaitez créer.
- Les colonnes (column1, column2, column3, ...) sont les attributs de la table.
- `datatype` spécifie le type de données de chaque colonne.

Création d'une table

Supposons que nous souhaitons créer une table pour stocker des informations sur les clients. Voici un exemple de création de cette table :

```
CREATE TABLE Clients
(
    ID INTEGER,
    Nom VARCHAR(50),
    Prénom VARCHAR(50),
    Email VARCHAR(100),
    DateNaissance DATE
);
```

Dans cet exemple, nous avons créé une table appelée "Clients" avec cinq colonnes : ID, Nom, Prénom, Email et DateNaissance. Les types de données des colonnes sont INT (entier), VARCHAR (chaîne de caractères variable) et DATE (date).

Contraintes de clé primaire

Il est courant d'inclure une contrainte de clé primaire lors de la création d'une table. Une clé primaire garantit que chaque enregistrement dans la table est unique. Voici comment vous pouvez définir une clé primaire :

```
CREATE TABLE Clients
(
    ID INTEGER PRIMARY KEY,
    Nom VARCHAR(50),
    Prénom VARCHAR(50),
    Email VARCHAR(100),
    DateNaissance DATE
);
```

Dans cet exemple, la colonne "ID" est définie comme clé primaire. Cela signifie que chaque ID doit être unique dans la table.

SELECT

Le **SELECT** est utilisé pour **récupérer des données** à partir de la **base de données**.

Exemple de requête SQL avec SELECT :

```
SELECT nom, prenom FROM employes;
```

L'exemple montre comment sélectionner les colonnes "nom" et "prenom" de la table "employes".

Syntaxe

```
SELECT colonne1, colonne2
FROM table
```

La requête SQL ci-dessus permet de sélectionner des données spécifiques dans une table en précisant les noms des colonnes souhaitées.

Exemples d'utilisation

```
SELECT nom, prenom
FROM employes
```

Cette requête permet de récupérer les noms et prénoms de tous les employés présents dans la table "employes".

LIMIT et OFFSET

- **LIMIT**: restreint le nombre de résultats renvoyés
- **OFFSET**: décale le point de départ des résultats

```
SELECT nom, prenom
FROM employes
LIMIT 5 OFFSET 3
```

L'utilisation de LIMIT et OFFSET est pratique pour la pagination des résultats à afficher sur une interface utilisateur.

WHERE

Le **WHERE** est utilisé pour **filtrer** les résultats.

```
SELECT * FROM table
WHERE condition;
```

Colonne 1	Colonne 2
Info 1	Info 2
Info 3	Info 4

- Exemple de conditions :
  - champ = valeur
  - champ <> valeur (différent de)
  - champ > valeur (supérieur à)
  - champ < valeur (inférieur à)
  - champ >= valeur (supérieur ou égal à)
  - champ <= valeur (inférieur ou égal à)

WHERE est un outil essentiel pour effectuer des requêtes spécifiques et pour limiter le nombre de résultats retournés.

Syntaxe

```
SELECT colonne1, colonne2
FROM table
WHERE condition
```

Cette syntaxe de base permet de sélectionner des colonnes spécifiques dans une table en fonction d'une condition. N'oubliez pas d'adapter les noms de colonne et de table en fonction de vos besoins.

Exemples d'utilisation

```
SELECT nom, prenom
FROM employes
WHERE salaire > 2000
```

Cette requête SQL permet d'afficher le nom et le prénom des employés ayant un salaire supérieur à 2000.

Manipulation de données

INSERT INTO

Syntaxe

Pour insérer une ou plusieurs lignes dans une table, utilisez la syntaxe suivante :

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

Il est important de préciser les colonnes et les valeurs correspondantes dans le même ordre pour éviter les erreurs.

Exemples d'utilisation

```
INSERT INTO employes (id, nom, prenom, salaire)
VALUES (1, 'Doe', 'John', 50000);
```

```
INSERT INTO employes (id, nom, prenom, salaire)
VALUES (2, 'Smith', 'Jane', 55000),
       (3, 'Brown', 'Michael', 60000);
```

Cette slide montre comment insérer des données dans une table avec la commande **INSERT INTO**. Le premier exemple ajoute un employé et le deuxième exemple ajoute deux employés en une seule instruction.

Insérer une ou plusieurs lignes

Vous pouvez insérer plusieurs lignes en une seule commande en ajoutant plusieurs **groupes de valeurs**.

```
INSERT INTO employes (id, nom, prenom, salaire)
VALUES (4, 'Martin', 'Paul', 45000),
       (5, 'Garcia', 'Maria', 48000);
```

Les données doivent correspondre à la liste des colonnes spécifiée entre parenthèses.

UPDATE

Syntaxe

Pour mettre à jour des données dans une table, utilisez la syntaxe suivante :

```
UPDATE table
SET colonne1 = nouvelle_valeur1, colonne2 = nouvelle_valeur2, ...
WHERE condition;
```

- **table** : le nom de la table à mettre à jour.
- **colonne1, colonne2, ...** : les colonnes à mettre à jour avec de nouvelles valeurs.
- **nouvelle_valeur1, nouvelle_valeur2, ...** : les nouvelles valeurs à attribuer aux colonnes spécifiées.
- **WHERE condition** : la condition qui détermine quelles lignes de la table doivent être mises à jour. Cette clause est optionnelle, mais si elle est omise, toutes les lignes de la table seront mises à jour.

Exemples d'utilisation

Mettre à jour le salaire d'un employé :

```
UPDATE employes
SET salaire = 55000
WHERE nom = 'Doe';
```

Ce code met à jour le salaire de l'employé dont le nom est 'Doe' dans la table `employes`.

Mettre à jour plusieurs colonnes :

```
UPDATE employes
SET salaire = 60000, prenom = 'John'
WHERE nom = 'Smith';
```

Cet exemple met à jour à la fois le salaire et le prénom de l'employé dont le nom est 'Smith'.

Mettre à jour plusieurs lignes

Vous pouvez également mettre à jour plusieurs lignes en utilisant une condition plus générale. Par exemple, pour augmenter le salaire de tous les employés de plus de 5 ans d'ancienneté :

```
UPDATE employes
SET salaire = salaire * 1.1
WHERE anciennete > 5;
```

Dans cet exemple, tous les employés ayant plus de 5 ans d'ancienneté verront leur salaire augmenté de 10 %.

Demo

```sql
CREATE TABLE if NOT EXISTS Clients
(
    ID INTEGER PRIMARY KEY AUTOINCREMENT,
    Nom VARCHAR(50),
    Prénom VARCHAR(50),
    Email VARCHAR(100),
    DateNaissance DATE
);

INSERT INTO Clients (Nom, Prénom, Email, DateNaissance)
VALUES
    ('AUBRY', 'Zenza', 'kenza@mail.fr', 1996);

SELECT ID, Prénom FROM Clients;
SELECT * FROM Clients ORDER BY Prénom ASC;
```