Base de données

SGBD - SQL

Yannick CHISTEL

Lycée Dumont d'Urville - CAEN

22 novembre 2020

Qu'est-ce qu'un SGBD ?

Définition

SGBD est l'acronyme de Système de Gestion de Base de Données.

Un SGBD est un logiciel qui permet de créer et gérer des bases de données accessibles à un ou plusieurs utilisateurs.

Le langage utilisé dans les SGBD est le SQL, acronyme de Structured Query Language.

Propriétés

Les fonctionnalités d'un SGBD sont :

- créer des bases de données, ensemble de relations (tables) ;
- créer des relations (tables) à partir d'un schéma relationnel ;
- ajouter, modifier, supprimer des données dans les relations (tables) ;
- effectuer des requêtes pour interroger les relations, obtenir des données, triées ou non, respectant une ou plusieurs conditions ;
- spécifier les contraintes d'intégrités : clefs primaires, clefs étrangères, domaine de valeurs ;
- sécuriser et pérenniser les données ;
- gérer les droits, permissions, des utilisateurs de la base de données.

SGBD

Principe

Un SGBD est un logiciel installé (souvent) sur un serveur. Dans ce mode de fonctionnement, l'accès aux données à plusieurs utilisateurs est possible.
Cela implique que le SGBD permette :

1. de créer, supprimer des utilisateurs, gérer l'authentification (login, mot de passe) pour la connexion à une base de données ;
2. de gérer les permissions en lecture et écriture sur les relations de la base de données.
3. de gérer les accès concurrents, c'est à dire autoriser la lecture et l'écriture des données à plusieurs utilisateurs en même temps.

Libre ou propriétaire

Il existe de nombreux SGBD qu'on peut classer en deux catégories : libre ou propriétaire.

1. Parmi les logiciels libres : MySQL, PostgreSQL, MariaDB ;
2. Parmi les logiciels propriétaires : ORACLE et Microsoft SQL Server

SQLITE

SQLITE est un SGBD qui a la particularité de s'installer sur toute plateforme en mode dit embarqué (non serveur). Cela permet de gérer des bases de données sur une machine personnelle, très utile pour le développement.

Structured Query Language

Définition

Le langage majoritairement utilisé dans les SGBD est le SQL : Structured Query Language.

Ce langage permet de réaliser des requêtes sur les relations d'une base de données pour en extraire les données qu'elles contient.

Le langage SQL utilise des clauses qui permettent :

- Pour la sélection de données, on utilise la clause SELECT. Elle peut être accompagnée des clauses DISTINCT, WHERE et ORDER BY pour affiner la recherche ;
- L'ajout de données dans une relation avec la clause INSERT INTO ;
- La mise à jour ou la modification d'une donnée avec la clause UPDATE ;
- La suppression d'un enregistrement d'une relation avec la clause DELETE.

Remarque

1. Une requête SQL se termine par un point-virgule. Dans un client, la saisie du point virgule déclenche la requête.
2. Une requête SQL peut s'écrire sur plusieurs lignes.
3. Les clause présentées ci-dessus sont des clauses importantes mais il en existe beaucoup d'autres qui permettent de créer des relation, supprimer des relations, etc.

Clause SELECT ... FROM

Syntaxe

1. La clause SELECT réalise la sélection de données dans une relation :

   SELECT attribut1, attribut2,... **FROM** relation ;

2. Cette clause peut s'accompagner de la clause DISTINCT qui évite la redondance des données lorsque la requête renvoie plusieurs fois les mêmes valeurs.

   SELECT DISTINCT attribut1, attribut2,... **FROM** relation ;

3. La sélection de tous les attributs d'une table peut se faire avec l'astérisque * :

   SELECT * FROM relation ;

4. La sélection peut être triée par ordre croissant ou décroissant avec la clause ORDER BY :

   SELECT * FROM relation **ORDER BY** attribut1 **DESC**, attribut2 **ASC** ;

Exemple

1. ```
   SELECT titre,ann_pub FROM 'livre';
   ```
2. ```
   SELECT DISTINCT nom,prenom FROM 'auteur';
   ```
3. ```
   SELECT * FROM 'auteur';
   ```
4. ```
   SELECT nom,ann_naiss FROM 'auteur' ORDER BY ann_naiss DESC, nom ASC;
   ```

Clause SELECT ... FROM ... WHERE

Syntaxe

La clause WHERE impose une condition à respecter dans la requête :

SELECT attribut1, attribut2,... **FROM** relation **WHERE** condition ;

Opérateurs utilisés dans les conditions :

opérateur	**Description**
=	égal
<> ou !=	différent
> et <	supérieur à et inférieur à
>= et <=	supérieur ou égal à et inférieur ou égal à
IN	Liste de plusieurs valeurs possibles
BETWEEN	Valeur comprise dans un intervalle donné
LIKE	Recherche en spécifiant le début, milieu ou fin d'un mot.
IS NULL	Valeur est nulle
IS NOT NULL	Valeur n'est pas nulle

Remarque

Plusieurs conditions sont possibles avec les opérateurs AND et OR ;
La négation d'une condition se fait avec NOT.

Exemples avec la clause WHERE

Exemple

1. On recherche les noms et prénoms des auteurs nés en 1920 :

```
SELECT nom,prenom FROM 'auteur' WHERE ann_naiss = 1920;
```

2. On recherche les noms et prénoms des auteurs nés entre 1900 et 2000 :

```
SELECT nom,prenom FROM 'auteur' WHERE ann_naiss BETWEEN 1900 AND 2000;
```

3. On recherche les titres des livres qui commencent par le mot "La" :

```
SELECT titre,ann_pub FROM 'livre' WHERE titre LIKE 'La%';
```

4. On recherche les noms et prénoms des auteurs contenus dans une liste :

```
SELECT nom,prenom FROM 'auteur' WHERE nom IN ('Asimov','Orwell');
```

5. On recherche les titres des livres dont l'année de publication est supérieure à 1960 et inférieure à 1980 :

```
SELECT titre FROM 'livre' WHERE ann_pub>1960 AND ann_pub<1980;
```

Fonctions d'agrégation

Statistiques

Les fonctions d'agrégation permettent d'effectuer quelques statistiques sur les tables d'une base de données. Les principales fonctions sont les suivantes :

- AVG() pour calculer la moyenne des valeurs d'un attribut ;
- COUNT() pour compter le nombre d'enregistrements sur une table ou un un attribut précisé ;
- MAX() pour récupérer la valeur maximum d'un attribut sur un ensemble de ligne. Cela s'applique à la fois pour des données numériques ou alphanumériques ;
- MIN() pour récupérer la valeur minimum de la même manière que MAX() ;
- SUM() pour calculer la somme de valeurs sur un ensemble d'enregistrements.

Exemple

```
SELECT COUNT(titre) FROM 'film';
=> renvoie le nombre de titres de la relation film
SELECT MAX(duree) FROM 'film';
=> renvoie la durée du film le plus long
SELECT MIN(duree) FROM 'film';
=> renvoie la durée du film le plus court
SELECT AVG(duree) FROM 'film';
=> renvoie la durée moyenne des films de la relation
SELECT SUM(duree) FROM 'film' WHERE realisateur=3;
=> renvoie la durée totale des films du réalisateur n°3
```

Clause JOIN ... ON

Syntaxe

La clause JOIN permet de rassembler deux ou plusieurs relations à l'aide d'attributs de chaque relation qui ont des valeurs égales. On utilise en général les clefs étrangères des relations.

SELECT attribut1, attribut2,... **FROM** relation 1

JOIN relation2 ON attribut_relation_1 = attribut_relation_2

JOIN relation3 ON attribut_relation_2 = attribut_relation_3 ... ;

On peut également ajouter la clause WHERE aux clauses JOIN.

Exemple

1. On réalise une jointure entre les tables **livre**, **livre_theme** et **theme** :

```
SELECT DISTINCT titre FROM livre
JOIN livre_theme ON id_livre=livre_id
JOIN theme ON theme_id=id_theme;
```

2. On recherche les titres des livres dont le thème est science fiction :

```
SELECT titre FROM 'livre'
JOIN 'livre_theme' ON livre_id=id_livre
JOIN 'theme' ON theme_id=id_theme
WHERE theme='science fiction';
```

Clause INSERT INTO ... VALUES

Syntaxe

La clause INSERT INTO permet d'ajouter un enregistrement à une relation.

INSERT INTO relation (attribut_1,..., attribut_n) **VALUES** (valeur_1,..., valeur_n) ;

Exemple

1. Insertion de 2 livres dans la relation livre :

```
INSERT INTO 'livre' VALUES (17,'Vingt mille lieues sous les mers',1870,10);
INSERT INTO 'livre' VALUES (18,'Dôme',2009,10) ;
```

2. Insertion d'un auteur dans la relation auteur :

```
INSERT INTO 'auteur' VALUES (11,'King','Stephen',1947,1);
```

Remarque

1. Il n'est pas obligatoire de préciser les attributs mais il faut bien veiller à mettre les valeurs dans l'ordre des attributs de la relation.
2. On peut mettre les attributs dans un ordre différent en prenant soin de mettre les valeurs dans le même ordre.

Clause UPDATE et DELETE

Syntaxe

1. La clause UPDATE permet de mettre à jour une relation en modifiant des valeurs.

   UPDATE relation **SET** attribut = nouvelle valeur **WHERE** attribut = ancienne valeur ;

2. La clause **DELETE** supprime un enregistrement :

   DELETE FROM relation **WHERE** attribut = valeur ;

Exemple

1. Modification de la clef étrangère de la table livre avec la bonne valeur :

```
UPDATE 'livre' SET auteur_id=11 WHERE titre='Dôme';
```

2. Suppression de l'enregistrement de la table livre de clef primaire 17 :

```
DELETE FROM 'livre' WHERE id_livre=17;
```

Attention

1. Si la clause WHERE n'est pas précisée ou mal conditionnée dans un UPDATE, alors toutes les valeurs de l'attribut de la relation peuvent être modifiées.
2. Si la clause WHERE n'est pas donnée dans un DELETE, alors tous les enregistrements de la table sont supprimés.