Chapitre 6 : Le Langage SQL

Objectifs :

- 1. Maîtriser le langage SQL pour la définition des données (création, modification, suppression de tables, expression de contraintes d'intégrité)
- 2. Maîtriser le langage SQL pour la manipulation des données (interrogation de la base, ajout, mise à jour, et suppression d'enregistrements)
- 3. Maîtriser le langage de contrôle de données (privilèges et droits d'accès)

Durée : 3h Cours +1.5h TD

Eléments de contenu

Chapitre 6 : Le Langage SQL

I- Introduction

Le langage SQL (Structured Query Langage : langage de requêtes structurées) est un langage non procédural, conçu par IBM dans les années 70. SQL est basée sur l'algèbre relationnelle (opérations ensemblistes et relationnelles). Normalisé dès 1986, SQL constitue le standard aux bases de données relationnelles. Ce langage permet l'accès aux données et se compose de quatre sous-ensembles :

- Le Langage de Définition de Données : LDD (Data DefinitionLanguage DDL) : Ce langage permet la définition et la mise à jour de la structure de la base de données (tables, attributs, vues, index, ...).
- Le Langage d'Interrogation de Données : LID : Ce langage permet de rechercher des informations utiles en interrogeant la base de données. Certains considèrent ce langage comme étant une partie du LMD.
- Le Langage de Manipulation de Données : LMD (Data Manipulation Language DML) : Ce langage permet de manipuler les données de la base et de les mettre à jour.
- Le Langage de Contrôle de Données : LCD (Data Control Language DCL) : Ce langage permet de définir les droits d'accès pour les différents utilisateurs de la base de données, donc il permet de gérer la sécurité de la base et de confirmer et d'annuler les transactions.

II- Expression des contraintes

1- Valeur NULL

Un attribut qui n'est pas renseigné, et donc vide, est dit contenir la valeur 'NULL'. Cette valeur n'est pas zéro, c'est une absence de valeur.

2- Contraintes d'intégrité

Une contrainte d'intégrité est une règle qui permet de contrôler la validité et la cohérence des valeurs entrées dans les différentes tables de la base. Elle peut être définie sous deux formes :

- Dans les commandes de création des tables.
- Au moment de la modification de la structure de la table.

Il existe des contraintes :

- Sur un attribut : La contrainte porte sur un seul attribut. Elle suit la définition de l'attribut.

Ces contraintes sont :

  - NOT NULL : Spécifie que pour toute occurrence, l'attribut doit avoir une valeur (la saisie de ce champ est obligatoire)

- UNIQUE : Toutes les valeurs de l'attribut sont distinctes.
- PRIMARY KEY : L'attribut est une clé primaire pour la table et elle peut être remplacée par UNIQUE et NOT NULL.
- REFERENCES table (attribut) : Il s'agit d'une contrainte d'intégrité fonctionnelle par rapport à une clé ; chaque valeur de l'attribut doit exister dans 21 la table dont l'attribut est référencé. On utilise cette contrainte pour les clés étrangères.
- CHECK : C'est une contrainte associée à une condition qui doit être vérifiée par toutes les valeurs de l'attribut (domaine des valeurs de l'attribut).

- Sur une table : La contrainte porte sur un ensemble d'attributs d'une même table, une virgule sépare la définition d'une contrainte de table des définitions des attributs. Ces contraintes sont :
  - UNIQUE (attri, attrj,…) : L'unicité porte sur le n-uplet des valeurs.
  - PRIMARY KEY (attri, attrj,…) : Clé primaire de la table (clé composée).
  - FOREIGN KEY (attri, attrj, …) REFERENCES table (attrm, attrn, …) [ON DELETE CASCADE] : Désigne une clé étrangère sur plusieurs attributs.

  L'option ON DELETE CASCADE indique que la suppression d'une ligne de la table de référence va entraîner automatiquement la suppression des lignes référencées.

Il est possible de donner un nom à une contrainte grâce au mot clé CONSTRAINT suivi du nom que l'on donne à la contrainte, de telle manière à ce que le nom donné s'affiche en cas de non-respect de l'intégrité, c'est-à-dire lorsque la clause que l'on a spécifiée n'est pas validée.

III- Le Langage de Définition des Données : LDD

SQL permet de définir des schémas de bases de données composées de tables et de vues.

1- Création d'une table

SQL permet de créer des relations sous forme de tables et de définir lors de la création des contraintes d'intégrité variées sur les attributs. Une commande de création permet donc de préciser le nom de la table et de définir les éléments de la table correspondant aux colonnes ou aux contraintes.

Syntaxe :

CREATE TABLE<nom de la table> (nom_colonne_1 type_colonne_1,

nom_colonne_2 type_colonne_2,….,

nom_colonne_ntype_colonne_n) ;

Exemple :

- AGENCE (NumAg, NomAg, VilleAg)

CREATE TABLE Agence(NumAg number(4) PRIMARY KEY, NomAg varchar(10),

VilleAg varchar(20));

- CLIENT (NumCl, NomCl, VilleCl)

CREATE TABLE Client (NumCl number (8) PRIMARY KEY, NomCl varchar (20),

VilleCl varchar(20));

- COMPTE (NumC, NumAg#, NumCl#, Solde)

CREATE TABLECompte (NumC number (20), NumAg number (4), NumCl number (8),

solde number (10), PRIMARY KEY (NumC, NumAg, NumCl), CONSTRAINT fk-ag FOREIGN KEY (NumAg) REFERENCES Agence, CONSTRAINT FOREIGN KEY (NumCl) REFERENCES Client);

2- Création d'une vue

Une vue est une table calculée (crée) à partir des tables de base grâce à une requête.

Exemple :

CREATE VIEW Agence1 AS SELECT NumAg, villeAg FROM Agence ;

Pour la supprimer :

DROP VIEW nom-vue ;

3- Création d'index

La création d'un index permet d'accélérer les recherches d'informations dans la base. La ligne est retrouvée instantanément si la recherche peut utiliser un index, sinon la recherche se fait séquentiellement. Une autre utilité de la création d'index est de garantir l'unicité de la clé en utilisant l'option UNIQUE.

Syntaxe :

CREATE [UNIQUE] INDEX nom_index

ON nom_table (Attr1[ASC/DESC], Attr2[ASC/DESC], …);

• L'option UNIQUE permet de définir la présence ou non de doublons pour les valeurs de l'attribut.

• Les options ASC/DESC permettent de définir un ordre de classement des valeurs présentes dans l'attribut.

4- Modification d'une table

La définition d'une table ou d'autres éléments du schéma dotés d'un nom peut être modifiée à l'aide de la commande ALTER.

a. Ajouter une ou plusieurs colonnes

Syntaxe :

ALTER TABLE<nom_de_la_table>ADD (nom_colonne1 type_colonne1,

nom_colonne2 type_colonne2,……,

nom_colonne n type_colonne n) ;

Exemple

ALTER TABLE Agence ADD (actif number (10)) ;

b. Modifier le type ou une autre propriété d'une colonne :

Syntaxe

ALTER TABLE nom_table MODIFY (nom_colonne nouveau_type et/ou nouvelle_propriété) ;

Exemple

ALTER TABLE Agence MODIFY (NomAg varchar(20));

5- Renommage une table

Syntaxe

RENAME nom_table TO nouveau_nom_table ;

Exemple

RENAME Professeur TO Enseignant ;

6- Suppressiond'une table

Syntaxe

DROP nom_table ;

IV- Le langage d'interrogation de données : LID

1- Syntaxe générale

La recherche d'information dans une base de données s'effectue à l'aide de la commande SQL SELECT. Dans cette section, nous présenterons une syntaxe simplifiée de l'instructionSELECT.

Syntaxe :

SELECT [ALL ou DISTINCT] attribut1, attribut2, …,attributN

FROM nom_de_la_table

WHERE condition

GROUP BY expression

HAVING condition

ORDER BY nom-attribut [ASC/DESC];

Remarque :

Le caractère ***** après **SELECT** indique que toutes les colonnes de la table doivent être prises en compte.

L'option **ALL** (par défaut) permet de sélectionner l'ensemble des lignes satisfaisant à la condition logique.

L'option **DISTINCT** permet de ne conserver que des lignes distinctes, en éliminant les doublons.

La clause **FROM** : permet de spécifier la (ou les) table (s) à interroger.

La clause **WHERE** : réalise l'opération de restriction de l'algèbre relationnelle en filtrant les lignes de la table qui répondent au critère de sélection.

Les clauses **GROUP BY** (grouper les résultats), **HAVING** (restriction avec condition) et **ORDER BY** (tri des résultats par ordre croissant ou décroissant) sont facultatives.

Exemple :

- SELECT NumAg, NomAg
  FROM Agence

  WHERE villeAg = 'Tunis;

Permet de sélectionner les agences (numéro et nom) de la ville de Tunis.

- SELECT AVG(solde)
  FROM Compte;

Permet de déterminer la moyenne des soldes de tous les comptes.

2- Les prédicats

c. BETWEEN / NOT BETWEEN :

Teste l'appartenance d'une valeur à un intervalle.

Exemple : n° des comptes dont le solde est compris entre 600 et 1000.

```
SELECT NumC
FROM Compte
WHERE solde BETWEEN 600 AND 1000;
```

d. LIKE :

Permet de faire une recherche approximative.

Le caractère % remplace un ensemble de caractères.

Le caractère – remplace un seul caractère.

Exemple : nom des clients dont la ville commence par A.

```
SELECT NomCl
FROM Client
WHERE villeCl LIKE 'A%';
```

e. IS NULL / IS NOT NULL :

Permet de tester si un champ a été affecté.

Exemple : nombre d'agences n'ayant pas un actif affecté ?

```
SELECT count (NumAg)
FROM Agence
WHERE actif IS NULL ;
```

3- La jointure

```
SELECT attribut1, attribut2,……
FROM table1 [alias], table2 [alias],……
WHERE table1.attribut= table2.attribut;
```

Exemple: donner le total de soldes des comptes des clients de la ville de Sousse.

```
SELECT sum(solde)
FROM Compte C, Client Cl
WHERE Cl.villeCl = 'Sousse'
AND Cl.NumCl=C.NumCl;
```

4- Les requêtes imbriquées

a. IN / NOT IN:

Permet de tester la présence ou l'absence d'une valeur particulière dans un ensemble de valeurs.

Exemple : nom des clients de la ville Sousse ou Tunis ?

```
SELECT NomCl
FROM Client
WHERE VilleCl IN ('Sousse,'Tunis') ;
```

b. ALL\ANY :

ALL: Compare chacune des valeurs de l'ensemble à une valeur particulière et retourne vrai si la comparaison est évaluée pour chacun des éléments.

ANY : Compare chacune des valeurs de l'ensemble à une valeur particulière et retourne vrai si la comparaison est évaluée pour au moins un des éléments.

Exemple : n° agence ayant au moins un comptedont le solde est strictement supérieur à chacun des comptes de l'agence A1 ?

```
SELECT NumAg
FROM Compte C
WHERE solde> ALL (SELECT solde
                  FROM Compte C
                  WHEREC.NumAg='A1');
```

c. EXISTS / NOT EXISTS :

Retourne vrai si une requête imbriquée retourne au moins une ligne.

Exemple : Intitulé des agences ayant au moins un compte rouge ?

```
SELECT NomAg
FROM Agence A
WHERE EXISTS (SELECT NumAg
              FROM Compte C
              WHERE A. NumAg=C.NumAg
```

And C.solde<0) ;

V- Le langage de manipulation des données : LMD

1- L'insertion de tuples

L'ajout de tuples se fait à l'aide de la commande INSERT. Il faut fournir le nom d'une relation et une liste de valeurs pour le tuple. Les valeurs doivent être écrites dans le même ordre que celui dans lequel ont été spécifiés les attributs auxquels elles correspondent lors de la création de la table avec CREATE TABLE.

Syntaxe

Syntaxe 1 :

INSERT INTO nom_de_la_table VALUES (valeur_attribut1, valeur_attribut2,…., valeur_attributN);

Syntaxe 2 :

INSERT INTO nom_de_la_table (attribut1, attribut3, attributN)VALUES(valeur_attribut1, valeur_attribut3, valeur_attributN);

L'utilisation de la Syntaxe2 nécessite que :

- Les autres attributs dans lesquels on n'a pas inséré des valeurs ne doivent pas être accompagnés par la contrainte NOT NULL lors de la création de la table.

Exemple :

- INSERT INTO Client VALUES (08954365, 'Mohamed Ali, 'Tunis');
- INSERT INTO Agence (NumAg, NomAg, VilleAg) VALUES (0045, 'Ag Rades', Rades');

2- La mise à jour de tuples

La commande UPDATE sert à modifier les valeurs des attributs d'un ou plusieurs tuples sélectionnés.

Syntaxe :

UPDATE nom_de_la_table

SET attribut1 = nouvelle_valeur1, …, attributN = nouvelle_valeurN

WHERE condition ;

Remarque

La clause WHERE sélectionne les tuples à modifier.

La clause SET spécifie les attributs à modifier et leurs nouvelles valeurs.

Exemple :

UPDATE Employe SET Salaire = 950 WHERE Matricule = 'IngMEC01' ;

3- La suppression de tuples

La commande DELETE supprime des tuples d'une relation. La clause WHERE sert à sélectionner les tuples à supprimer.

Syntaxe :

DELETE FROM nom_de_la table WHERE condition ;

VI- Le langage de contrôle des données : LCD

Plusieurs personnes peuvent travailler simultanément sur une base de données. Toutefois, ces personnes n'ont pas forcément les mêmes besoins : certaines peuvent par exemple nécessiter de modifier des données dans la table, tandis que les autres ne l'utiliseront que pour la consulter. Ainsi, il est possible de définir des permissions pour chaque personne en leur octroyant un mot de passe.

Cette tâche incombe à l'administrateur de la base de données (en anglais DBA, DataBase Administrator). Il doit dans un premier temps définir les besoins de chacun, puis les appliquer à la base de données sous forme de permissions. Le langage SQL permet d'effectuer ces opérations grâce à deux clauses :

- GRANT permet d'accorder des droits à un (parfois plusieurs sur certains SGBD) utilisateur
- REVOKE permet de retirer des droits à un (ou plusieurs sur certains SGBD) utilisateur

Les permissions (appelées aussi droits ou privilèges) peuvent être définies pour chaque (un grand nombre) clause. D'autre part, il est aussi possible de définir des rôles c'est-à-dire de permettre à d'autres utilisateurs d'accorder des permissions.

1- La clause GRANT

GRANT : Permet au propriétaire d'une table ou vue de donner à d'autres utilisateurs des droits d'accès à celles-ci.

Syntaxe :

GRANT Liste_Privilège ON Table/ Vue TO Utilisateur [WITH GRANT OPTION];

Les privilèges sont :

- SELECT : droit de lecture
- INSERT : droit d'insertion de lignes
- UPDATE : droit de modification de lignes
- DELETE : droit de suppression de lignes
- ALTER : droit de modification de la structure de la table
- INDEX : droit de création d'index
- ALL : Tous les droits

Exemple :

GRANT SELECT ON Client TO User1;

2- La clause REVOKE

Un utilisateur ayant accordé un privilège peut l'annuler à l'aide de la commande REVOKE.

Syntaxe :

REVOKE Liste_Privilège ON Table/Vue FROM Utilisateur;

Exemple:

REVOKE SELECT ON Produit FROM User1;