Langage SQL

Département d’informatique

Introduction

- Le langage SQL (*Structured Query Language*) peut être considéré comme le langage d'accès normalisé aux bases de données.
- Il est aujourd'hui supporté par la plupart des produits commerciaux que ce soit par les systèmes de gestion de bases de données micro tel que *Access* ou par les produits plus professionnels tels que *Oracle*.
- Il a fait l'objet de plusieurs normes ANSI/ISO dont la plus répandue aujourd'hui est la norme SQL3 qui a été définie en 1999.
- Le succès du langage SQL est dû essentiellement à sa simplicité et au fait qu'il s'appuie sur le schéma conceptuel pour énoncer des requêtes en laissant le SGBD responsable de la stratégie d'exécution.

Introduction

- Le langage SQL propose un langage de requêtes ensembliste.
- Le langage SQL ne possède pas la puissance d'un langage de programmation (entrées/sorties, instructions conditionnelles, boucles et affectations, etc,)
- Pour certains traitements il est donc nécessaire de coupler le langage SQL avec un langage de programmation plus complet.
- De manière synthétique, on peut dire que SQL est un langage relationnel, il manipule donc des tables par l'intermédiaire de requêtes qui produisent également des tables.

Historique

- En 1970, E.F. CODD, directeur de recherche du centre IBM de San José, invente le modèle relationnel qui repose sur une algèbre relationnelle.
- En 1977, création du langage SEQUEL (Structured English Query Language) et mise en place du Système R, prototype de base de données reposant sur la théorie de CODD.
- En 1981, la société ORACLE CORP lance la première version de son système de gestion de base de données relationnelle (SGBDR), IBM sort SQL/DS et RTI lance INGRES.
- En 1982, IBM sort SQL/DS pour son environnement VM/CMS et l'ANSI lance un projet de normalisation d'un langage relationnel.
- En 1983, IBM lance DB2.
- En 1986, la société SYBASE lance son SGBDR conçu selon le modèle Client-Serveur.

Historique

- La première norme SQL (SQL-1) de l'ISO apparaît. Il existe désormais plusieurs dizaines de produits proposant le langage SQL et tournant sur des machines allant des micros aux gros systèmes.
- Les différents produits phares ont évolué, la norme SQL est passée à SQL-2, puis SQL-3. SQL est désormais un langage incontournable pour tout SGBD moderne.
- *Oracle* et *Informix* dominent le marché actuel, *SQL-Server* (de Microsoft) tente de s'imposer dans le monde des PC sous NT. À côté des ces produits, très chers, existent heureusement des systèmes libres et gratuits : *MySQL* et *PostgreSQL* sont les plus connus.

Catégories d'instructions

- Les instructions SQL sont regroupées en catégories en fonction de leur utilité et des entités manipulées. Nous pouvons distinguer quatre catégories:
  1. La définition des éléments d'une base de données (tables, colonnes, clefs, index, contraintes, …),
  2. La manipulation des données (insertion, suppression, modification, extraction, …),
  3. La gestion des droits d'accès aux données (acquisition et révocation des droits),
  4. La gestion des transactions.

Langage de définition de données

- Le **langage de définition de données** (LDD, ou *Data Definition Language)* est un langage orienté au niveau de la structure de la base de données.
- Le LDD permet de créer, modifier, supprimer des objets. Il permet également de définir le domaine des données (nombre, chaîne de caractères, date, booléen, …) et d'ajouter des contraintes de valeur sur les données. Il permet enfin d'autoriser ou d'interdire l'accès aux données et d'activer ou de désactiver l'audit pour un utilisateur donné.
- Les instructions du LDD sont : CREATE, ALTER, DROP, AUDIT, NOAUDIT, ANALYZE, RENAME, TRUNCATE.

Langage de manipulation de données

- Le **langage de manipulation de données** (LMD, ou *Data Manipulation Language)* est l'ensemble des commandes concernant la manipulation des données dans une base de données. Le LMD permet l'ajout, la suppression et la modification de lignes, la visualisation du contenu des tables et leur verrouillage.
- Les instructions du LMD sont : INSERT, UPDATE, DELETE, SELECT, EXPLAIN, PLAN, LOCK TABLE.

Autres Langages

- **Langage de protections d'accès** (*Data Control Language*) s'occupe de gérer les droits d'accès aux tables.
  - Les instructions du DCL sont : GRANT, REVOKE.
- **Langage de contrôle de transaction** (*Transaction Control Language*) gère les modifications faites par le LMD, c'est-à-dire les caractéristiques des transactions et la validation et l'annulation des modifications.
  - Les instructions du TCL sont : COMMIT, SAVEPOINT, ROLLBACK, SET TRANSACTION
- Le **SQL intégré** (*Embedded SQL*) permet d'utiliser SQL dans un langage de Haut niveau tel que C, Java, Cobol, etc.
  - Les instructions du SQL intégré sont : DECLARE, TYPE, DESCRIBE, VAR, CONNECT, PREPARE, EXECUTE, OPEN, FETCH, CLOSE, WHENEVER.

CREATE TABLE

La commande CREATE TABLE permet de créer une table en SQL. Un tableau est une entité qui est contenu dans une base de données pour stocker des données ordonnées dans des colonnes. La création d'une table sert à définir les colonnes et le type de données qui seront contenus dans chacun des colonne (entier, chaîne de caractères, date, valeur binaire …).

CREATE TABLE nom_de_la_table

(

colonne1 type_donnees [CONSTRAINT C1] contrainte_1,

colonne2 type_donnees [CONSTRAINT C2] contrainte_2,

colonne3 type_donnees [CONSTRAINT C3] contrainte_3,

colonne4 type_donnees [CONSTRAINT C4] contrainte_4,

[CONSTRAINT T1] contrainte_T_1,….

)

Les types de données

- **INTEGER :** Entiers signés codés sur 32 bits.
- **BIGINT :** Entiers signés codés sur 64bits.
- **REAL :** Réels comportant 6 chiffres significatifs codés sur 32 bits.
- **DOUBLE PRÉCISION :** Réels comportant 15 chiffres significatifs codés sur 64 bits.
- **NUMERIC[(précision, [longueur])] :** Données numériques à la fois entières et réelles avec une précision de 1000 chiffres significatifs. longueur précise le nombre maximum de chiffres significatifs stockés et précision donne le nombre maximum de chiffres après la virgule.
- **CHAR(longueur) :** Chaînes de caractères de longueur fixe inférieure à 255
- **VARCHAR(longueur) :** Chaînes de caractères de longueur variable inférieure à 2000
- **DATE :** Données constituées d'une date.
- **TIMESTAMP :** Données constituées d'une date et d'une heure.
- **BOOLEAN :** Valeurs Booléenne.
- **MONEY :** Valeurs monétaires.
- **TEXT :** Chaînes de caractères de longueur variable.

Contraintes de colonne

Les différentes contraintes de colonne que l'on peut déclarer sont les suivantes :

NOT NULL / NULL : Interdit /autorise (NULL) l'insertion de valeur NULL pour cet attribut.

UNIQUE : Désigne l'attribut comme clé secondaire de la table.

PRIMARY KEY : Désigne l'attribut comme clé primaire de la table. La clé primaire étant unique, cette contrainte ne peut apparaître qu'une seule fois dans l'instruction. La définition d'une clé primaire composée se fait par l'intermédiaire d'une contrainte de table.

REFERENCES table [(colonne)] [ON DELETE CASCADE] : Contrainte d'intégrité référentielle pour l'attribut de la table en cours de définition. Les valeurs prises par cet attribut doivent exister dans l'attribut colonne qui possède une contrainte PRIMARY KEY ou UNIQUE dans la table table. En l'absence de précision d'attribut colonne, l'attribut retenu est celui correspondant à la clé primaire de la table table spécifiée.

CHECK (condition) :Vérifie lors de l'insertion de n-uplets que l'attribut réalise la condition « condition ».

DEFAULT valeur : Permet de spécifier la valeur par défaut de l'attribut.

Contraintes de table

Les différentes contraintes de table que l'on peut déclarer sont les suivantes :

PRIMARY KEY (colonne, …) :Désigne la concaténation des attributs cités comme clé primaire de la table. Cette contrainte ne peut apparaître qu'une seule fois dans l'instruction.

UNIQUE (colonne, …) :Désigne la concaténation des attributs cités comme clé secondaire de la table. Dans ce cas, au moins une des colonnes participant à cette clé secondaire doit permettre de distinguer le n-uplet. Cette contrainte peut apparaître plusieurs fois dans l'instruction.

FOREIGN KEY (colonne, …) REFERENCES table [(colonne, …)] [ON DELETE CASCADE | SET NULL]

Contrainte d'intégrité référentielle pour un ensemble d'attributs de la table en cours de définition. Les valeurs prises par ces attributs doivent exister dans l'ensemble d'attributs spécifié et posséder une contrainte PRIMARY KEY ou UNIQUE dans la table table.

CHECK (condition) : Cette contrainte permet d'exprimer une condition qui doit exister entre plusieurs attributs de la ligne.

DROP TABLE

La commande DROP TABLE en SQL permet de supprimer définitivement une table d'une base de données. Cela supprime en même temps les éventuels index, trigger, contraintes et permissions associées à cette table.

Attention :

- Il faut utiliser cette commande avec attention car une fois supprimée, les données sont perdues.
- Avant de l'utiliser sur une base importante il peut être judicieux d'effectuer un backup (une sauvegarde) pour éviter les mauvaises surprises.

DROP TABLE Table

ALTER TABLE

La commande ALTER TABLE en SQL permet de modifier une table existante. Il est ainsi possible d'ajouter une colonne, d'en supprimer une ou de modifier une colonne existante, par exemple pour changer le type.

Ajouter une colonne

ALTER TABLE nom_table
ADD nom_colonne type_donnees

Supprimer une colonne

ALTER TABLE nom_table
DROP nom_colonne

Renommer une colonne

ALTER TABLE nom_table
CHANGE ancien_nom nouveau_nom type_donnees

Modifier une colonne

ALTER TABLE nom_table
MODIFY nom_colonne type_donnees

Renommer une Table

ALTER TABLE nom_table
RENAME TO nouveau_nom

INSERT INTO

- L'insertion de données dans une table s'effectue à l'aide de la commande INSERT INTO.
- Cette commande permet au choix d'inclure une seule ligne à la base existante ou plusieurs lignes d'un coup.
- Pour insérer des données dans une base, il y a 2 syntaxes principales :
  - Insérer une ligne en indiquant les informations pour chaque colonne existante (en respectant l'ordre)

    INSERT INTO table **VALUES** ('valeur 1', 'valeur 2', ...)
  - Insérer une ligne en spécifiant les colonnes que vous souhaiter compléter.

    INSERT INTO table (nom_colonne_1, nom_colonne_2, ...)

    VALUES ('valeur 1', 'valeur 2', ...)
  - Il est possible d'insérer une ligne en renseigner seulement une partie des colonnes

UPDATE

La commande UPDATE permet de modifier les valeurs d'une ou plusieurs colonnes, dans une ou plusieurs lignes existantes d'une table.

UPDATE nom_table

SET nom_col_1 = {expression_1},

nom_col_2 = {expression_2},

...

nom_col_n = {expression_n}

WHERE (Condition)

Les valeurs des colonnes nom_col_1, nom_col_2,… nom_col_n sont modifiées dans toutes les lignes qui satisfont la condition. En l'absence d'une clause WHERE, toutes les lignes sont mises à jour. Les expressions expression_1, expression_2,… expression_n peuvent faire référence aux anciennes valeurs de la ligne.

SELECT

- L’utilisation la plus courante de SQL consiste à lire des données issues de la base de données.
- Cela s’effectue grâce à la commande SELECT, qui retourne des enregistrements dans un tableau de résultat.
- Cette commande peut sélectionner une ou plusieurs colonnes d’une table.

SELECT [Ch_1, Ch_2,…, Ch_N]

FROM [Table_1,Table_2,…,Table_M]

Cas spécifique:

SELECT * FROM Table

DISTINCT

- L’utilisation de la commande SELECT en SQL permet de lire toutes les données d’une ou plusieurs colonnes.
- Cette commande peut potentiellement afficher des lignes en doubles.
- Pour éviter des redondances dans les résultats il faut simplement ajouter DISTINCT après le mot SELECT.

SELECT DISTINCT Colonne

FROM Table

WHERE

La commande WHERE dans une requête SQL permet d'extraire les lignes d'une base de données qui respectent une condition.

SELECT [Ch1, Ch2,…, Chn]

FROM [Table1,Table2,…,Tablen]

WHERE [Condition1] **AND/OR** [Condition2]…

Opérateurs de comparaisons

=, <>, !=, <, >, >=, <=

IN : Liste de plusieurs valeurs possibles

BETWEEN : Valeur comprise dans un intervalle donnée

LIKE : Recherche en spécifiant le début, milieu ou fin d'un mot (%, _)

IS NULL/ IS NOT NULL : Valeur est nulle / Valeur n'est pas nulle

WHERE

SELECT colonne

FROM table

WHERE colonne **IN (** valeur1, valeur2, valeur3, ... **)**

SELECT colonne

FROM table

WHERE colonne **BETWEEN** 'valeur1' **AND** 'valeur2‘

SELECT colonne

FROM table

WHERE colonne **LIKE “modele”**

SELECT colonne

FROM table

WHERE colonne **IS NULL**

SELECT colonne

FROM table

WHERE colonne **IS NOT NULL**

GROUP BY

- La commande GROUP BY est utilisée en SQL pour grouper plusieurs résultats et utiliser une fonction de totaux sur un groupe de résultat.
- Sur une table qui contient toutes les ventes d'un magasin, il est par exemple possible de regrouper les ventes par clients identiques et d'obtenir le coût total des achats pour chaque client.

SELECT colonne1, fonction(colonne2)

FROM table

GROUP BY colonne1

HAVING

La condition HAVING en SQL est presque similaire à WHERE à la seule différence que HAVING permet de filtrer en utilisant des fonctions telles que SUM(), COUNT(), AVG(), MIN() ou MAX().

SELECT colonne1, fonction(colonne2)

FROM Table

GROUP BY colonne1

HAVING fonction(colonne2) operateur valeur

Operateurs

=, <>, !=, <, >, >=, <=, IN, BETWEEN

ORDER BY

- La commande ORDER BY permet de trier les lignes dans un résultat d'une requête SQL.
- Il est possible de trier les données sur une ou plusieurs colonnes, par ordre ascendant ou descendant.

SELECT colonne1, colonne 2, colonne 3

FROM Table

ORDER BY [list of colonne]

SELECT colonne1, colonne2, colonne3

FROM table

ORDER BY colonne1 **DESC**, colonne2 **ASC**

AS (alias)

- Dans le langage SQL il est possible d'utiliser des **alias** pour renommer temporairement une colonne ou une table dans une requête.
- Cette astuce est particulièrement utile pour faciliter la lecture des requêtes.

SELECT colonne1 **AS** C1, colonne2 **AS** C2

FROM Table

LIMIT et OFFSET

- La clause LIMIT est à utiliser dans une requête SQL pour spécifier le nombre maximum de résultats que l'ont souhaite obtenir.
- Cette clause est souvent associé à un OFFSET, c'est-à-dire effectuer un décalage sur le jeu de résultat.

```
SELECT *
FROM Table
LIMIT 5

SELECT *
FROM Table
LIMIT 5  OFFSET 10
```

UNION

- La commande UNION de SQL permet de mettre bout-à-bout les résultats de plusieurs requêtes utilisant elles-même la commande SELECT.
- C'est donc une commande qui permet de concaténer les résultats de 2 requêtes ou plus.
- Pour l'utiliser il est nécessaire que chacune des requêtes à concaténer retournes le même nombre de colonnes, avec les mêmes types de données et dans le même ordre.

SELECT colonne 1, colonne 2

FROM Table 1

UNION

SELECT colonne 1, colonne 2

FROM Table 2

INTERSECT

- La commande SQL INTERSECT permet d'obtenir l'intersection des résultats de 2 requêtes.
- Cette commande permet donc de récupérer les enregistrements communs à 2 requêtes.
- Cela peut s'avérer utile lorsqu'il faut trouver s'il y a des données similaires sur 2 tables distinctes.

SELECT colonne 1, colonne 2

FROM Table 1

INTERSECT

SELECT colonne 1, colonne 2

FROM Table 2

EXCEPT

- La commande EXCEPT dans SQL s'utilise entre 2 instructions pour récupérer les enregistrements de la première instruction sans inclure les résultats de la seconde requête.
- Si un même enregistrement devait être présent dans les résultats des 2 syntaxes, ils ne seront pas présent dans le résultat final.

SELECT colonne 1, colonne 2

FROM Table 1

EXCEPT

SELECT colonne 1, colonne 2

FROM Table 2

Ordre des commandes

SELECT *
FROM Table
WHERE condition
GROUP BY expression
HAVING condition
{ **UNION** | **INTERSECT** | **EXCEPT** }
ORDER BY expression
LIMIT count
OFFSET start

EXISTS

La commande EXISTS dans SQL s'utilise dans une clause conditionnelle pour savoir s'il y a une présence ou non de lignes lors de l'utilisation d'une sous-requête.

Remarque : cette commande n'est pas à confondre avec la clause **IN**. La commande EXISTS vérifie si la sous-requête retourne un résultat ou non, tandis que **IN** vérifie la concordance d'une à plusieurs données.

```
SELECT colonne1
FROM Table1
WHERE EXISTS (SELECT colonne2
              FROM Table2
               WHERE colonne3 = Valeur
              )
```

ALL

- La commande ALL dans SQL permet de comparer une valeur dans l'ensemble de valeurs d'une sous-requête.
- En d'autres mots, cette commande permet de s'assurer qu'une condition est « égale », « différente », « supérieure », « inférieure », « supérieure ou égale » ou « inférieure ou égale » pour **tous** les résultats retourné par une sous-requête.

```
SELECT colonne1
FROM Table1
WHERE condition1 > ALL ( SELECT colone2
                         FROM Table2
                         WHERE condition2
                       )
```

ANY / SOME

- La commande ANY (ou SOME) dans SQL permet de comparer une valeur avec le résultat d'une sous-requête.
- Il est ainsi possible de vérifier si une valeur est « égale », « différente », « supérieur », « supérieur ou égale », « inférieur » ou « inférieur ou égale » pour **au moins une des valeurs** de la sous-requête.

```
SELECT colonne1
FROM Table1
WHERE condition1 > ANY ( SELECT colone2
                         FROM Table2
                         WHERE condition2
                       )
```

Quelques fonctions sur les chaines

- **CONCAT**(Valeur 1, Valeur 2) : permet de fusionner deux chaines
- **LENGTH**(chaine de caractères) : permet de retourner la longeur d'une chaine
- **REPLACE**(chaine de caractères, chaine a remplacer, chaine remplaçante)
- **SUBSTRING**(chaine de caractères, debut, longueur) : permet de retourner une sous-chaine
- **REVERSE**(chaine de caractères): permet d'inverser une chaine
- **UPPER(**chaine de caractères**) / LOWER(**chaine de caractères**) :** majuscule /minuscule
- **LOCATE**(chaine rechercher, chaine source): cherhcer une chaine dans une autre chine

Autres fonctions utiles

- **ROUND** (valeur**,** précision) : permet d'arrondir un résultat numérique
- **RAND()**: permet de sélectionner un nombre aléatoire dans l'intervalle [0,1]
- **NOW()**: permet de retourner la date et l'heure du système
- **DATEDIFF**(date 1, date 2): permet de retourner le nombre de jours entre les deux dates
- **MONTH**(date): permet de retourner le mois
- **YEAR**(date): permet de retourner l'année
- **CAST**(valeur **AS** type) : permet d'effectuer un changement de type