Chapitre 3

SQL Avancé (LDD,LMD et LCD)

3.1 Présentation de SQL

SQL signifie « Structured Query Language » c'est-à-dire « Langage d'interrogation structuré ». En fait SQL est un langage complet de gestion de bases de données relationnelles. Il est introduit par IBM dans les années 70, il est considéré comme l'évolution du langage SEQUEL, il est commercialisé tout d'abord par ORACLE. SQL est devenu le langage standard des systèmes de gestion de bases de données (SGBD) relationnelles (SGBDR).
C'est à la fois :

- Un langage d'interrogation de la base *(SELECT).*
- Un langage de manipulation des données *(UPDATE, INSERT, DELETE).*
- Un langage de définition des données *(CREATE, ALTER, DROP).*
- Un langage de contrôle de l'accès aux données *(GRANT, REVOKE).*
- Un langage de gestion de transactions *(COMMIT, ROLLBACK).*

SQL a été normalisé dès 1986 mais les premières normes, trop incomplètes, ont été ignorées par les éditeurs de SGBD : (ANSI : American National Standard Institute)

- SQL 1 initial : ANSI date de 1986.
- SQL 1 intégrité référentielle : ANSI date de 1989.
- SQL 2 ANSI date de 1992, extension de SQL1.
- SQL 3 ANSI date de 1999. (appelée aussi SQL99) est la nouvelle norme SQL avec l'intégration du modèle objet.

3.2 Définition de données

SQL est un Langage de Définition des Données (LDD), il permet de créer des tables dans une base de données relationnelle, ainsi que d'en modifier ou en supprimer.
Ordres pour la création et la suppression de la base de donnes :

Créer une Bases de données
CREATE DATABASE nom_bdd ;

Supprimer une Bases de données
DROP DATABASE nom_bdd ;

3.2.1 Création de tables

La commande CREATE TABLE crée la définition d'une table

Syntaxe :

CREATE TABLE nom_table (- - définition des colonnes
Col1 TYPE (taille) [NOT NULL [UNIQUE]]
[DEFAULT valeur]
[PRIMARY KEY]
[REFERENCES table]
[CHECK condition],
...,
- - contraintes de table
[PRIMARY KEY (liste de colonnes)],
[UNIQUE (liste de colonnes)],
... ,
[FOREIGN KEY (liste de colonnes) REFERENCES table
[ON DELETE {RESTRICT | CASCADE | SET NULL}]
[ON UPDATE {RESTRICT | CASCADE | SET NULL}],

... ,
[CHECK condition],
...
) ;

Principaux types de données

- le type CHAR pour les colonnes qui contiennent des chaînes de longueur constante (CHAR(n)).
- le type VARCHAR pour les colonnes qui contiennent des chaînes de longueurs variables (VARCHAR(n)).
- le type SMALLINT Nombres entiers sur 2 octets.
- le type INTEGER Nombres entiers sur 4 octets.
- le type DECIMAL(n,m) correspond à des nombres décimaux qui ont n chiffres significatifs et m chiffres après la virgule.
- le type DATE réserve 2 chiffres pour le mois et le jour et 4 pour l'année.
- le type TIME pour les heures, minutes et secondes.
- ...

Contraintes d'intégrité

- *NOT NULL* : valeur null impossible.
- *UNIQUE* : interdit qu'une colonne contienne deux valeurs identiques.
- *PRIMARY KEY* : définit la clé primaire de la table.
- *FOREIGN KEY* : indique que la colonne que l'on définit est une clé étrangère qui fait référence à la colonne de la table tableref (contrainte d'intégrité référentielle).
- *CHECK* : donne une condition que les colonnes de chaque ligne devront vérifier (plage ou liste de valeurs).
- *CASCADE* : cascader les suppressions ou Modifications.
- *SET NULL* : rendre nul les attributs référençant.
- *RESTRICT* : rejet de la mise à jour c'est l'option par défaut.

Les contraintes peuvent s'exprimer :

- Soit au niveau colonne (contraintes locales) : valables pour une colonne.

— Soit au niveau table (contraintes globales) : valables pour un ensemble de colonnes d'une table.

Les contraintes se définissent :

— Soit lors de la création des tables, dans l'ordre CREATE TABLE ;

— Soit après la création des tables, par l'ordre ALTER TABLE permettant certaines modifications de la structure des tables.

EXEMPLE :

```
CREATE TABLE Clients (
idClient CHAR(6) PRIMARY KEY ,
nom VARCHAR(30) NOT NULL,
adresse VARCHAR(30) ,
numéroTelephone INTEGER
) ;

CREATE TABLE Produit (
idProduit CHAR(6) PRIMARY KEY ,
nom VARCHAR(30) NOT NULL,
marque VARCHAR(30) NOT NULL ,
prix DECIMAL(6,2) ,
- - contrainte de table
CHECH (marque IN ( BMW, TOYOTA,PEUGEOT) )
) ;

CREATE TABLE Vente (
idVente CHAR(6) PRIMARY KEY ,
referenceProduit CHAR(6) ,
idClient CHAR(6) ,
date DATE NOT NULL ,
- - contrainte de table
FOREIGN KEY (referenceProduit) REFERENCES Produit(idProduit) ON DELETE
CASCADE,
FOREIGN KEY (idClient) REFERENCES Clients (idClient) ON DELETE CASCADE,
```

) ;

3.2.2 Modification du schéma

Il est possible de supprimer ou de modifier la structure d'une table à l'aide des commandes :
DROP TABLE et ALTER TABLE

Renommer une table
ALTER TABLE nom_table RENAME TO nouveau_nom ;

Ajout ou modification de colonne
ALTER TABLE nom_table {ADD/MODIFY} ([nom_colonne type [contrainte], ...]) ;

Renommer une colonne
ALTER TABLE nom_table RENAME COLUMN ancien_nom TO nouveau_nom ;

Supprimer une colonne
ALTER TABLE nom_table DROP COLUMN nom_colonne ;

Ajout d'une contrainte de table
ALTER TABLE nom_table ADD [CONSTRAINT nom_contrainte]contrainte ;

Supprimer des contraintes
ALTER TABLE nom_table DROP CONSTRAINT nomContrainte ;

Suppression de données uniquement
TRUNCATE TABLE nom_table ;

3.2.3 Création d'index

Afin d'améliorer les temps de réponses, SQL propose un mécanisme d'index. Un index est associé à une table, mais il est stocké à part. Un index peut ne porter que sur une colonne (index simple) ou sur plusieurs (index multiple). Chaque valeur d'index peut ne désigner qu'une et une seule ligne, on parlera alors d'index unique donc de clef primaire, dans le cas contraire on parlera d'index dupliqué.

Remarque :
Un index est automatiquement créé lorsqu'une table est créée avec la contrainte PRIMARY KEY.

Création
CREATE INDEX index_nom ON table ;

- *index sur une seule colonne :*
  CREATE INDEX index_nom ON table (colonne1) ;
- *index sur plusieurs colonnes :*
  CREATE INDEX index_nom ON table (colonne1, colonne2) ;

Suppression
DROP INDEX nom_index ;

3.3 Manipulation de données

SELECT, INSERT, UPDATE et DELETE sont les 4 commandes de manipulation des données en SQL.

3.3.1 Modifier une base de données

Les commandes INSERT, UPDATE et DELETE permet la modification dune base de données.

— *La commande INSERT :*

La commande INSERT permet d'ajouter de nouvelles lignes à une table

Syntaxe

INSERT INTO NomTable (colonne1,colonne2,colonne3,...)

VALUES (Valeur1,Valeur2,Valeur3,...) ;

Insertion par une sélection

INSERT INTO NomTable (colonne1,colonne2,...)

SELECT colonne1,colonne2,... FROM NomTable2

WHERE condition

Exemple :

INSERT INTO Clients (idClient, nom, adresse, numéroTelephone) VALUES (c214, hamiche, cité Hamadi N114, 0654874125) ;

Remarque :

Les valeurs inconnues prennent la valeur NULL

— *La commande UPDATE :*

La commande UPDATE permet de changer des valeurs d'attributs de lignes existantes.

Syntaxe

UPDATE NomTable SET Colonne = Valeur Ou Expression

[WHERE condition]

Exemple :

UPDATE Clients SET numéroTelephone = 05874521 WHERE idClient = c214 ;

Remarque :

L'absence de clause WHERE signifie que les changements doivent être appliqués à toutes les lignes de la table cible.

— *La commande DELETE :*

La commande DELETE permet d'enlever des lignes dans une table.

Syntaxe

DELETE FROM NomTable

[WHERE condition]

Exemple :

DELETE FROM Clients WHERE idClient = 'c214' ;

Remarque :

L'absence de clause WHERE signifie que toutes les lignes de la table cible sont enlevées

3.3.2 Interroger une base de données

La commande SELECT permet de rechercher des données à partir de plusieurs tables ; le résultat est présenté sous forme d'une table réponse.

Syntaxe de base
SELECT [ALL|DISTINCT] NomColonne1,... | *
FROM NomTable1,...
WHERE Condition

- ALL : toutes les lignes.
- DISTINCT : pas de doublons.
- * : toutes les colonnes.
- La clause AS : SELECT Compteur AS Ctp FROM Vehicule

Expression des restrictions
Les conditions peuvent faire appel aux opérateurs suivants :

- Opérateurs logiques : AND, OR, NOT, XOR
- Comparateurs de chaînes : IN, BETWEEN, LIKE
- Opérateurs arithmétiques : +, -, /, %
- Comparateurs arithmétiques : =, <, >, >=, <=, <>
- Fonctions numérique : abs, log, cos, sin, mod, power,...
- Fonctions sur chaine : length, concat,...

Exemple
SELECT * FROM produit WHERE (prix>1000) AND (prix <=3000)
SELECT * FROM produit WHERE prix BETWEEN 1000 AND 3000
SELECT * FROM produit WHERE Marque IN ('TOYOTA','BMW')

Différentes clauses de SELECT

Syntaxe
SELECT *
FROM table
WHERE condition
GROUP BY expression
HAVING condition
{ UNION | INTERSECT | EXCEPT }
ORDER BY expression
LIMIT count

— *GROUP BY*
GROUP BY est utilisé pour grouper plusieurs résultats sur un groupe de résultat.
Exemple
SELECT COUNT (*)
FROM produit
GROUP BY marque

— *HAVING*
La clause HAVING permet de spécifier une condition de restriction des groupes. Elle sert à éliminer certains groupes, comme WHERE sert à éliminer des lignes.
Exemple
SELECT COUNT (*)
FROM produit
GROUP BY marque
HAVING COUNT (*) $>$ 40

— *ORDER BY*
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 (ASC|DESC).
Exemple

SELECT *
FROM produit
ORDER BY marque DESC

— *UNION*

UNION permet de concaténer les résultats de deux 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.

Exemple

SELECT marque FROM produit
UNION
SELECT marque FROM Vente

— *INTERSECT*

INTERSECT permet d'obtenir l'intersection des résultats de deux requêtes (récupérer les enregistrements communs à 2 requêtes). Cela permet de trouver s'il y a des données similaires sur 2 tables distinctes.

Exemple

SELECT marque FROM produit
INTERSECT
SELECT marque FROM Vente

— *Expression des jointures*

Le produit cartésien s'exprime simplement en incluant plusieurs tables après la clause FROM. La condition de jointure est exprimée après WHERE

Exemple

SELECT P.IdProduit, P.nom
FROM produit P , vente V
WHERE P.IdProduit = V.referenceProduit

— *Requête imbriquée*

SQL permet l'imbrication de sous-requêtes au niveau de la clause WHERE. Les sous-requêtes sont utilisées :

- dans des prédicats de comparaison : (=, <>, <, <=, >, >=)
- dans des prédicats : IN, EXISTS, ALL ou ANY.

Exemple

```
SELECT C.nom
FROM client C
WHERE idClient IN (
SELECT V.IdClient
FROM vente V
WHERE referenceProduit IN (
SELECT idProduit
FROM produit P
WHERE P.nom = 'p1' ))
```

— *Le prédicat EXISTS*

Il permet de tester si le résultat d'une sous-requête est vide ou non.

Exemple

```
SELECT P.nom
FROM produit P
WHERE NOT EXISTS ( SELECT *
FROM vente V
WHERE V.referenceProduit = P.IdProduit )
```

— *Le prédicat ALL ou ANY*

Ils permettent de tester si un prédicat de comparaison est vrai pour tous (ALL) ou au moins un (ANY) des résultats d'une sous-requête.

Exemple

```
SELECT C.nom
FROM client C, vente V, produit P
WHERE C.idClient = V. idClient AND P.idProduit = V.referenceProduit AND
P.prix >= ALL
( SELECT PR.prix
FROM client CL, vente VT, produit PR
WHERE CL.idClient = VT. idClient AND PP.idProduit = VT.referenceProduit
AND PR.nom = 'c1' )
```

3.4 Contrôle de Données

3.4.1 Contrôle des accès concurrents

La notion de transaction :
Une transaction est une unité logique de traitement qui est soit complètement exécutée, soit complètement abandonnée. Une transaction fait passer la BD d'un état cohérent à un autre état cohérent. Une transaction est terminée : soit par COMMIT, soit par ROLLBACK.

La commande COMMIT
La commande COMMIT termine une transaction avec succès ; toutes les mises à jour de la transaction sont validées. On dit que la transaction est validée.

La commande ROLLBACK
La commande ROLLBACK termine une transaction avec échec ; toutes les mises à jour de la transaction sont annulées (tout se passe comme si la transaction n'avait jamais existé). On dit que la transaction est annulée.

3.4.2 Contrôle des droits d'accès

La commande GRANT
La commande GRANT permet de passer des droits d'accès à un utilisateur ou un groupe d'utilisateurs

Syntaxe
GRANT privilèges ON table TO bénéficiaire

[WITH GRANT OPTION]
Les privilèges qui peuvent être passés sont :

- soit ALL (tous les privilèges)
- soit une liste de privilèges parmi :
  - SELECT
  - INSERT
  - UPDATE [(liste de colonnes)] : l'omission de la liste de colonnes signifie toutes les colonnes
  - DELETE

  Le bénéficiaire peut être :

- soit PUBLIC (tous les utilisateurs)
- soit un utilisateur ou un groupe d'utilisateurs

L'option WITH GRANT OPTION permet de passer un privilège avec le droit de le transmettre.

La commande REVOKE
La commande REVOKE permet de retirer des droits à un utilisateur ou groupe d'utilisateurs.

Syntaxe
REVOKE privilèges ON table FROM bénéficiaire

3.4.3 Notion de sous-schéma

L'objet VUE
Une vue est une table virtuelle (aucune implémentation physique de ses données) calculée à partir des tables de base par une requête.

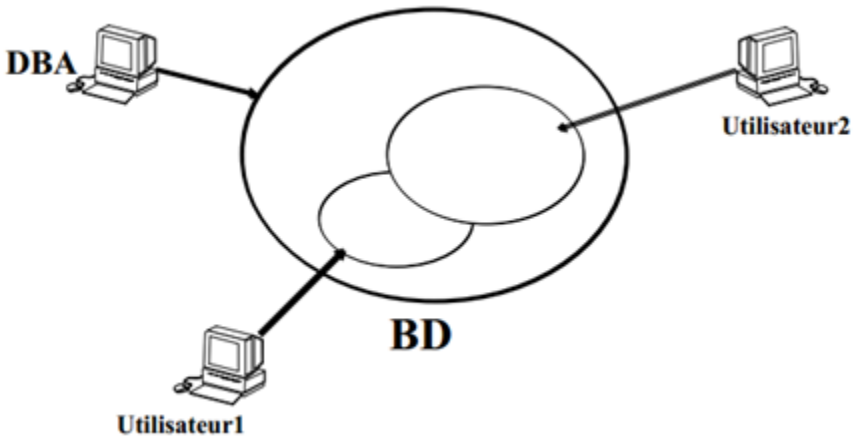

FIGURE 3.1 – Notion de Sous-Schéma

Une vue apparaît à l'utilisateur comme une table réelle, cependant les lignes d'une vue ne sont pas stockées dans la BD (La définition de la vue est enregistrée dans le DD). Les vues assurent l'indépendance logique, elles peuvent être utilisées pour cacher des données sensibles, ou pour montrer des données statistiques.

Création et suppression d'une VUE :

Création :

```
CREATE VIEW NomVue(NomColonne1,...) AS
SELECT NomColonne1,..
FROM NomTable
WHERE Condition
```

Suppression :

```
DROP VIEW nom_vue;
```

RENOME :

```
RENAME VIEW nom_vue TO nouveau_nom;
```

Intérêt des vues

— *Indépendance logique*
Le concept de vue permet d'assurer une indépendance des applications vis-à-vis des modifications du schéma (Assurer l'indépendance du schéma externe).

— *Simplification d'accès*
Les vues simplifient l'accès aux données en permettant par exemple une prédéfinition des jointures et en masquant ainsi à l'utilisateur l'existence de plusieurs tables (Création de résultats intermédiaires pour des requêtes complexes).
Exemple : La vue qui calcule les moyennes générales pourra être consultée par la requête :
SELECT * FROM Moyennes

— *Confidentialité des données*
Une vue permet d'éliminer des lignes sensibles et/ou des colonnes sensibles dans une table de base (éviter de divulguer certaines informations).

3.4.4 Rôles

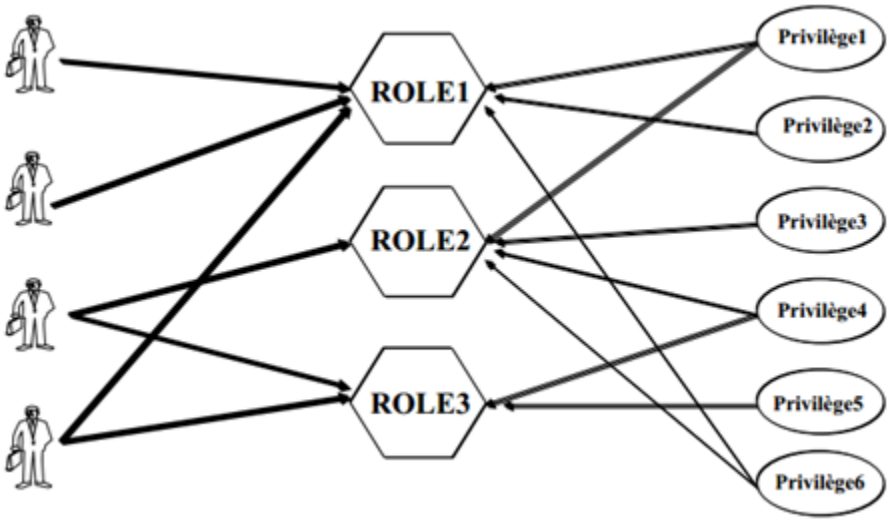

FIGURE 3.2 – Les Rôles

— Regroupement de privilèges pour des familles d'utilisateur

— Facilitent la gestion des autorisations des privilèges objet en évitant les ordres GRANT
— Un rôle par défaut est donné à un utilisateur
— Un utilisateur peut posséder plusieurs rôles mais il n'est connecté qu'avec un seul à la fois
— On peut donner un mot de passe pour certains rôles

Manipulation des rôles : Ordres

Création / Modification d'un rôle

{CREATE|ALTER} ROLE nom_role {NOT IDENTIFIED | IDENTIFIED {BY mot_de_passe|} EXTERNALLY} ;

Remplissage et attribution d'un rôle

GRANT {privilege1 | role1} TO nom_role ;
GRANT {privilege2 | role2} TO nom_role ;
GRANT ROLE nom_role TO user ;

Rôle par défaut ou activation

SET ROLE nom_role [IDENTIFIED BY mot_de_passe] ;

Suppression / Révocation d'un rôle

DROP ROLE nom_role ; REVOKE ROLE nom_role FROM user ;

Exemple

CREATE ROLE employeur ;
GRANT SELECT, INSERT,UPDATE(adresse, numéroTelephone) ON clients TO employeur ;

GRANT SELECT, INSERT ON produit TO employeur ;
GRANT ALL ON vente TO employeur ;
GRANT ROLE employeur TO Said, Salah, Karim ;

3.4.5 Contraintes évènementielles : Trigger

Tout comme les procédures stockées, les triggers servent à exécuter une ou plusieurs instructions. Mais à la différence des procédures, il n'est pas possible d'appeler un trigger : un trigger doit être déclenché par un événement. Un trigger est attaché à une table, et peut être déclenché par :

- Une insertion dans la table (INSERT).
- La suppression d'une partie des données de la table (DELETE) .
- La modification d'une partie des données de la table (UPDATE).

Par ailleurs, une fois le trigger déclenché, ses instructions peuvent être exécutées soit juste avant (BEFORE) l'exécution de l'événement déclencheur, soit juste après (AFTER).

Syntaxe
CREATE TRIGGER nom_trigger
{BEFORE | AFTER | INSTEAD OF}
{INSERT | DELETE | UPDATE [OF col1, col2,...]}
ON {nom_table | nom_vue}
[REFERENCING {NEW|OLD} AS <nom>]
[FOR EACH ROW]
[WHEN condition]
BLOC PL/SQL

- *BEFORE :* Exécution avant modification des données.
- *AFTER :* Exécution après modification des données.
- *INSTEAD OF :* Exécution à la place de l'ordre SQL envoyé.
- *INSERT, UPDATE, DELETE :* Action concernée par le déclencheur.

— *OLD :* représente les valeurs des colonnes de la ligne traitée avant qu'elle ne soit modifiée par l'événement déclencheur. Ces valeurs peuvent être lues, mais pas modifiées.
— *NEW :* représente les valeurs des colonnes de la ligne traitée après qu'elle a été modifiée par l'événement déclencheur. Ces valeurs peuvent être lues et modifiées.

Suppression des triggers

```
DROP TRIGGER nom_trigger;
```

Exemple 1 : Supprimer les produits correspondant au fournisseur supprimé (On suppose que chaque produit peut être livré par un seul fournisseur).

```
DELIMITER |
CREATE TRIGGER DeleteFournisseur
BEFORE DELETE ON Fournisseur
FOR EACH ROW
BEGIN
DELETE FROM Produit P WHERE P.idFournisseur = OLD.idFournisseur;
END |
DELIMITER;
```

Exemple 2 :

Contrôler l'existence d'un fournisseur lors de l'ajout d'un produit. Si pas de fournisseur, annuler la transaction.

```
DELIMITER |
CREATE TRIGGER InsertProduit
BEFORE INSERT ON Produit
FOR EACH ROW
BEGIN
WHEN ( NOT EXIST (
SELECT * FROM Fournisseur F WHERE F.IdFournisseur = NEW.IdFournisseur )
ABORT TRANSACTION;
END |
```

```
DELIMITER ;
```