

Chapitre 4 :

Le langage

SQL

I. INTRODUCTION

2

SQL est un acronyme pour “Structured Query Language” qui a été conçu par IBM. C’est maintenant le langage le plus utilisé dans les SGBD-R. Le langage SQL est un langage de définition (LDD) et de manipulation (LMD) de bases de données relationnelles,

Il est a évolué de manière à pouvoir être utilisé en mode interactif (comme un langage de script), en mode procédural (on crée des programmes effectuant plusieurs taches dans un même traitement. Ex : un script Shell), ou intégré à un autre langage (bibliothèques SQL pour python, C, Java,...).

SQL permet de :

- définir les données (CREATE, ALTER, DROP)
- interroger la base et formuler des requêtes (SELECT)
- manipuler les données (INSERT, UPDATE, DELETE)
- contrôler l'accès aux données (GRANT, REVOKE)

II. Le Language de Manipulation des Données

3

A. langage de requête

```
SELECT [DISTINCT] nom-cols [AS  
nom-cols]  
FROM nom-tables  
[ WHERE conditions]  
[ GROUP BY nom-cols]  
[ HAVING conditions ]  
[ ORDER BY nom-cols [ASC | DESC] ]
```

1. Interrogation d'une seule table

Soit le schéma relationnel suivant:

Employé (N_Emp, Nom, Prénom,
DateNaissance, Region, Salaire,
#N_Dept, #N_sup)

Département (N_Dept, NomD, #Directeur)

Projet (N_Projet, NomP, Lieu, #N_Dept)

Travaille (#N_Emp, #N_Pro, Heures)

a. PROJECTION

5

- ▶ Afficher une table entièrement

SELECT *

FROM Nom_table;

- ▶ Exemple :

- ▶ Tous les departement ?

- ▶ SQL : **SELECT ***

FROM Departement;

a. PROJECTION

6

```
SELECT Nom_Col1, ..., Nom_ColN  
FROM Nom_table;
```

► Exemple :

Requête : Liste des noms de Departement ?

```
SQL : SELECT NomD  
FROM Departement;
```

La clause DISTINCT permet d'éliminer les doublons.

b. SELECTION

7

```
SELECT Nom_Colonne  
FROM Nom_Table  
WHERE critère ;
```

► Exemple :

► Requête : La liste des employes habitant à Dakar,

► SQL :

```
SELECT N_Emp  
FROM Employe  
WHERE Region='Dakar';
```

c. Opérateurs de comparaison

8

► =, >, <, >=, <=, <> (ou !=)

► In et not in

► AND , OR

► Requête : quels sont les employés (N_Emp, Nom , prenom) qui travaillent au departement 12 qui habitent thies, dakar ou St louis et qui ont un salaire superieur à 300 000 ?

► SQL :

SELECT N_Emp, Nom , prenom

FROM Employe

WHERE N_Dept=12 AND Salaire>300000 AND (region='Dakar' OR region='Thies' OR region='St louis') ;

c. Opérateurs de comparaison

9

► =, >, <, >=, <=, <> (ou !=)

► In et not in

► AND , OR

► Requête : quels sont les employés (N_Emp, Nom , prenom) qui travaillent au departement 12 qui habitent thies, dakar ou St louis et qui ont un salaire superieur à 300 000 ?

SQL : OU

SELECT N_Emp, Nom , prenom

FROM Employe

**WHERE N_dept= 12 AND Salaire>300000 AND Region
in ('Dakar','Thies','St louis') ;**

c. Opérateurs de comparaison

10

- ▶ LIKE : appartenance à une chaîne de caractères
 - ▶ '_' remplace n'importe quel caractère
 - ▶ '%' remplace n'importe quelle chaîne de caractères
- ▶ Exemple 1:
 - ▶ Requête : Quels sont les noms de projet qui commencent par Sen?
 - ▶ SQL :

```
SELECT NomP
FROM projet
WHERE NomP like 'Sen%';
```
- ▶ Exemple 2:
 - ▶ Requête : Quels sont les noms de projet possédant un 'a' en seconde position ?
 - ▶ SQL :

```
SELECT NomP
FROM projet
WHERE NomP like '_a%';
```

c. Opérateurs de comparaison

11

- ▶ BETWEEN : appartenance à un intervalle

- ▶ Exemple :

- ▶ Requête : Quels sont les salariés gagnant entre 200 000 et 500 000 ?

- ▶ SQL : **SELECT N_Emp, Nom, Prenom, Salaire**

FROM Employe

**WHERE Salaire BETWEEN 200000 AND
500000;**

c. Opérateurs de comparaison

12

- ▶ **IS NULL et IS NOT NULL**

- ▶ **Exemple :**

- ▶ Requête : quels sont les employés dont la date de naissance n'a pas été renseignée

- ▶ SQL : **SELECT N_Emp, Nom, Prenom**
FROM Employe
WHERE Datenaiss IS NULL;

d. Tri des résultats

13

► ORDER BY

SELECT attribut1, attribut2, ...

FROM Nom_table

ORDER BY attribut1 [ASC], attribut2 [DESC], ... ;

► Exemple :

- Requête : Les employes classés par département et du plus grand salaire au plus petit salaire

- SQL : SELECT *

FROM Employe

ORDER BY N_dept ASC, Salaire DESC;

e. Nom de colonne

14

- ▶ Les colonnes constituant le résultat d'un SELECT peuvent être renommées dans le SELECT
 - ▶ Requête : salaire de chaque employé
 - ▶ SQL : `SELECT Nom, Salaire « SALAIRE MENSUEL »
FROM Employe;`

f. Fonctions

15

- ▶ La relation résultat
 - ▶ ne comportera qu'une ligne
 - ▶ ou pourra simplement être considérée comme un nombre
- ▶ Fonctions numériques :
 - ▶ AVG : moyenne
 - ▶ Exemple : Salaire moyen des Employes
 - ▶ SQL : SELECT AVG(salaire)
 FROM Employe;
 - ▶ SUM : somme
 - ▶ Exemple : Masse salarial des employes de Kaolack:
 - ▶ SQL : SELECT SUM(salaire)
 FROM Employe
 WHERE Region = 'Kaolack';

f. Fonctions

16

► **COUNT** : nombre d'éléments sélectionnés

► Requête : Nombre de projet du department 15 ?

► SQL : **SELECT COUNT(n_projet)**

FROM projet

WHERE n_dept=15;

► **MIN** :Retourne le minimum

► Requête : Quel est l'employe qui a le plus petit salaire?

► SQL : **SELECT n_emp, nom, prenom, Min(Salaire)**

FROM Employe;

► **MAX**: Retourne le maximum

► Requête : salaire max ?

► SQL : **SELECT Max(Salaire)**

FROM Employe;

- ▶ ~~Expressions et fonctions sur les dates~~
 - ▶ Opérateurs sur les dates : + et -
 - ▶ `date +/- nombre` : le resultat est une date obtenue en ajoutant le nombre de jours `nombre` à la date `date`.
 - ▶ `date2 - date1` : le resultat est le nombre de jours entre les deux dates.

g. Regroupements

18

- ▶ Il est possible de subdiviser la table en groupes
- ▶ Permet d'appliquer les fonctions d'aggrégation à des sous-groupes
 - ▶ Requête : Donner le nombre d'employe par Region?
 - ▶ SQL : **SELECT Region, Count(*)**
FROM Employe
GROUP BY Region;

h.Selection des groupes¹⁹

- ▶ **HAVING** : conditions imposées aux groupes (de lignes) à sélectionner
 - ▶ pour éviter la confusion avec la clause WHERE (qui ne s'applique qu'à des lignes seules)
- ▶ Exemple :
 - ▶ Requête : Donner les regions ou la masse salariale superieur à 20 millions?
 - ▶ SQL : **SELECT Region, SUM(salaire)**
FROM Employe
GROUP BY Region,
HAVING Sum(salaire)>20000000;

i. Quantificateurs

20

- ▶ **ALL** + opérateur de comparaison : teste si une expression est vérifiée dans tous les cas de figure

 - ▶ Requête : Tous les employés sont-ils nés avant le 1er janvier 1983 ?
 - ▶ SQL : `'1983-01-01' > ALL (SELECT Datenaiss FROM Employe) ;`
- ▶ **SOME** ou **ANY** : expression vraie si la comparaison est vérifiée pour au moins une valeur
 - ▶ Requête : vérifiez si au moins un salarié est né après 2005
 - ▶ SQL : `'2005-01-01' < ANY (SELECT Datenaiss FROM Employe) ;`

2. Interrogations sur plusieurs tables :

SQL permet la liaison de plusieurs tables via 3 possibilités :

- ▶ Les opérations de **jointure** entre 2 tables en se basant sur l'égalité entre l'un des attributs de chaque table
- ▶ Le principe des **requêtes imbriquées** qui repose sur le fait que le résultat d'une requête est une table
- ▶ L'utilisation **d'opérations ensemblistes** pour combiner le résultat de plusieurs requêtes.

a. Jointures

22

SELECT ...

FROM ~~nom_table1, nom_table2...~~

WHERE critère;

- ▶ pas de condition de sélection : résultat obtenu = produit cartésien des tables présentes derrière le FROM.
- ▶ Requête : quelles sont les employés du department finance ?
- ▶ SQL :
SELECT N_emp, Nom, Prenom
FROM Employé, Département
WHERE Employe.N_dept =
Département.N_dept AND NomD= 'Finance' ;

b. Auto-jointure :

23

- ▶ Requête : Donner pour chaque employé le nom de son supérieur hiérarchique.
- ▶ **SQL :** **SELECT Employe.N_emp,
Employe.Prenom, Employe.Nom,
chef.N_emp, chef.Nom
FROM Employe, Employe chef
WHERE Employe.N_sup=
chef.N_emp;**

c. Autres jointures

24

Le critère d'égalité est le critère de jointure le plus naturel. Mais on peut utiliser d'autres types de comparaisons comme critères de jointures.

- ▶ Requête : Quels sont les employés gagnant plus que l'employé de matricule M10 ?
- ▶ SQL : `SELECT Employe.N_emp, Employe.nom, Employe.prenom, Employe.salaire`
- ▶ `FROM Employe, Employe Empbis`
`WHERE Employe.salaire > Empbis.salaire`
`AND Empbis.N_emp = 'M10';`

Application 1

25

On considère les relations suivantes :

Joueur (Nom, Prenom, Age, Nationalité)

Rencontre (NomGagnant, Nomperdant
, LieuTournoi, Annee, Score)

Gain (NomJoueur, LieuTournoi, Annee, Rang,
Prime, NomSponsor)

Sponsor (NomSponsor, LieuTournoi, Annee,
Adresse, MtContribution)

Application 1

26

Exprimez sur cette base de données les requêtes suivantes :

Q1 : Nom et primes des joueurs sponsorisés par Orange entre 2004 et 2009

Q2 : Nom et age des joueurs ayant participé au Tournoi de Roland Garros de 2006.

Q3 : Nom et Nationalité des joueurs ayant participé à la fois au Tournoi de Roland Garros et à celui de Wimbledon en 2006.

Q4 : Nom des joueurs ayant gagné une prime supérieur à 500000 en 2009

Q5 : Nom des joueurs ayant toujours perdu à Wimbledon

Q6: Nom des sponsor de Roland Garros depuis 2010 et les montants contribués

Q7 : Nom, Prénom, Age et Nationalité des joueurs ayant participé à tous les tournois de 2006.

3. requêtes imbriquées

27

Une **SOUS-REQUETE**, ou requête imbriquée, (sous requête interne, *en anglais : inner subquery*) est une requête utilisée pour **CONSTITUER UN JEU DE RESULTAT** qui va **SERVIR DANS UNE REQUETE PRINCIPALE**, la requête appelante (requête externe, *en anglais : outer query*),

- - en général comme **élément de comparaison** dans la clause **WHERE**,
- - parfois comme **valeur d'une colonne** dans la clause **SELECT**.

3. requêtes imbriquées

28

```
SELECT ... [, (sous requête) AS alias_colonne]  
FROM ...  
WHERE nom_colonne operateur (sous  
requête)  
[ GROUP BY ... ]  
[ HAVING nom_colonne operateur (sous  
requête) ]  
[ ORDER BY ... ]  
;
```

3. requêtes imbriquées

29

► Sous-interrogation ramenant une seule valeur

► Requête : Quels sont les employés qui sont dirigés par Mamadou fall ?

► SQL : **SELECT N_emp, Nom , prenom**
FROM Emp
WHERE Chef = (SELECT N_emp
FROM Emp
WHERE Nom ="fall" and
prenom="mamadou");

► Remarque : auto-jointure possible pour répondre à cette question

3. requêtes imbriquées

30

- ▶ une sous-interrogation qui ne ramène aucune ligne se termine avec un code d'erreur.
- ▶ une sous-interrogation ramenant plusieurs lignes provoquera aussi, dans ce cas, une erreur

3. requêtes imbriquées

31

- ▶ **Sous-interrogation ramenant plusieurs lignes**
- ▶ Avec des opérateurs de comparaison admettant à leur droite un ensemble de valeurs comme :
 - ▶ l'opérateur IN
 - ▶ les opérateurs obtenus en ajoutant ANY ou ALL à la suite d'un opérateur de comparaison classique (=, <>, >, >=, <, <=)

3. requêtes imbriquées

32

Exemple:

► Requête : Quels sont les employés gagnant plus que tous les employés du département 30 ?

► SQL : **SELECT N_emp, Nom, prenom, Salaire**

```
FROM Emp WHERE salaire >
ALL(SELECT Salaire
     FROM Emp
     WHERE N_Dep=30) ;
```

3. requêtes imbriquées

33

Sous-interrogation ramenant plusieurs colonnes

- ▶ Exemple : Quels sont les employés ayant la même fonction et le même supérieur que Fatou Ndiaye ?

- ▶ SQL : `SELECT N_emp, prenom, Nom,
Fonction, chef`

`FROM Emp`

`WHERE (Fonction, chef) = (SELECT
Fonction, chef FROM Emp`

`WHERE Nom = 'Ndiaye'
and Prenom="Fatou");`

- ▶ **Sous-interrogation ramenant au moins une ligne**
- ▶ L'opérateur **EXISTS** permet de construire un prédicat vrai si la sous-interrogation qui suit ramène au moins une ligne.
 - ▶ Requête : Quels sont les employés travaillant dans un département qui a procédé à des embauches depuis le début de l'année 2001 ?
 - ▶ SQL :

```
SELECT *  
FROM Emp Empbis  
WHERE EXISTS (SELECT *  
              FROM EMP  
              WHERE Embauche >= '01-jan-01'  
              AND N_Dept = Empbis.N_Dept);
```

4. Les opérateurs ensemblistes

- ▶ permettent de "joindre" des tables verticalement c'est-à-dire de combiner dans un résultat unique des lignes provenant de deux interrogations. Les opérateurs ensemblistes sont les suivants :
 - ▶ l'union : UNION
 - ▶ l'intersection : INTERSECT
 - ▶ la différence relationnelle : MINUS
- ▶ La syntaxe d'utilisation est la même pour ces trois opérateurs :
- ▶ `SELECT ... { UNION | INTERSECT | MINUS } SELECT ...`

Les opérateurs ensemblistes

- ▶ Dans une requête utilisant des opérateurs ensemblistes :
 - ▶ Tous les SELECT doivent avoir le même nombre de colonnes sélectionnées, et leur types doivent être un à un identiques.
 - ▶ Les doubles sont éliminés (DISTINCT implicite).
 - ▶ Les noms de colonnes sont ceux du premier SELECT.
- ▶ On peut combiner le résultat de plus de deux SELECT au moyen des opérateurs UNION, INTERSECT, MINUS.

SELECT ... UNION SELECT ... MINUS SELECT ...

- ▶ Expression évaluée de gauche à droite. Modification de l'ordre d'évaluation par des parenthèses.

SELECT ...

UNION (SELECT ...

MINUS

SELECT ...)

Les opérateurs ensemblistes

- ▶ Exemple : Lister tous les enseignants

```
SELECT Nom, Prénom
```

```
FROM MdC
```

```
UNION
```

```
SELECT Nom, Prénom
```

```
FROM Professeur ;
```

Comment interpréter une requête complexe

-
- ▶ **multitable ?**
 - on considère les tables spécifiées dans la clause **FROM**
 - on effectue la jointure de ces tables selon le critère de jointure de la clause **WHERE**
 - on sélectionne les lignes de la jointure sur la base des autres conditions de la clause **WHERE**
 - on classe ces lignes en groupes comme spécifié dans la clause **GROUP BY**
 - on ne retient que les groupes qui vérifient la clause **HAVING**
 - de chacun de ces groupes, on extrait les valeurs demandées dans la clause **SELECT**
 - les valeurs demandées sont ordonnées selon la clause **ORDER BY** éventuelle.

Comment interpréter une requête complexe

multitable ?

► Exemple :

```
SELECT N°Client, COUNT(*), SUM(QtéCom)
FROM Commande C, LigneCom L
WHERE C.N°Com = L.N°Com
AND N°Pro = 'PA 60'
GROUP BY N°Client
HAVING COUNT(*) >= 2
ORDER BY N°Client
```

III. Le Language de Manipulation des Données :

1. la modification de données

A. Insertion de nouveaux n-uplets

Syntaxe:

```
INSERT INTO nom_table(nom_col1, nom_col2,...)
```

```
VALUES (val1, val2...)
```

► Exemple :

```
INSERT INTO Emp (N_emp, Nom, Prénom, date_naiss, region,  
                salaire, N_dept, Chef)
```

```
VALUES(96035, 'Diop', 'Khadim', 1990-01-12, "Fatick", 100000, 12,  
      90053);
```

III. Le Language de Manipulation des Données :

1. la modification de données

B. Insertion de nouveaux n-uplets avec select

- Il est possible d'insérer dans une table des lignes provenant d'une autre table. La syntaxe est la suivante :

```
INSERT INTO nom_table(nom_col1, nom_col2, ...)
```

```
SELECT ...
```

- Exemple : Insérer dans la table Bonus les noms, prenom et salaires des directeurs.

```
INSERT INTO bonus
```

```
SELECT nom, prenom, salaire FROM emp WHERE fonction =  
'directeur';
```

III. Le Language de Manipulation des Données :

42

1. la modification de données

C. Modification de lignes

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

```
UPDATE nom_table
```

```
SET nom_col1 = {expression1 | ( SELECT ... ) },
```

```
nom_col2 = {expression2 | ( SELECT ... ) }
```

```
[WHERE critère];
```

- Exemple : Augmenter de 10% les salaires des ingénieurs.

```
UPDATE emp
```

```
SET salaire = salaire * 1.1
```

```
WHERE fonction = 'ingenieur' ;
```

III. Le Language de Manipulation des Données :

1. la modification de données

D. Suppression de lignes

La commande DELETE permet de supprimer des lignes d'une table.

Syntaxe :

```
DELETE FROM nom_table
```

```
WHERE critère;
```

- Toutes les lignes pour lesquelles le critère est évalué à vrai sont supprimées. En l'absence de clause WHERE, toutes les lignes de la table sont supprimées.

- Exemple :

```
DELETE FROM emp
```

```
WHERE fonction = 'retraité';
```

VI. Le langage de définition de données

1. Création d'une table

```
CREATE TABLE nom_table  
    (nom_col1 TYPE1,[NOTNULL/  
    PRIMARY KEY/FOREIGN KEY]  
    nom_col2 TYPE2,[.../.../...]  
...);
```

- ▶ Types acceptés :
- ▶ CHAR(longueur), VARCHAR(longueur)
- ▶ SMALLINT, INTEGER, DECIMAL(m,n), FLOAT, SERIAL(n)
- ▶ DATE

VI. Le langage de définition de données

45

1. Création d'une table

```
CREATE TABLE Departement  
  (N_Dep SERIAL(20) NOTNULL PRIMARY KEY,  
   NomDep VARCHAR(20),  
   Directeur SERIAL(20),  
   Budget DECIMAL(7,0));
```

```
CREATE TABLE Employé  
  (N_emp SERIAL(20) NOTNULL PRIMARY KEY,  
   Nom VARCHAR(20),  
   Prénom VARCHAR(20), DateNaissance DATE,  
   Region VARCHAR(80) DEFAULT "Dakar", salaire  
   DECIMAL(6,0), Num_Dept SERIAL(20) NOTNULL FOREIGN KEY  
   REFERENCES departement(N_dep);
```

VI. Le langage de définition de données

2. Suppression et modification d'une table

- ▶ DROP TABLE nom_table ;

- ▶ Exemple : DROP TABLE Etudiant ;

- ▶ Modification d'une table

- ▶ Ajout d'une ou plusieurs colonnes :

ALTER TABLE nom_table

ADD(nom_col1 TYPE1, nom_col2 TYPE2, ...);

- ▶ option : [BEFORE nom_col_before]

- ▶ Exemple : On aimerait connaître le téléphone des étudiants

ALTER TABLE Etudiant

ADD(Téléphone DECIMAL(10,0)

BEFORE NDep);

VI. Le langage de définition de données

2. Suppression et modification d'une table

- ▶ Suppression d'une colonne :

```
ALTER TABLE nom_table
```

```
DROP nom_col;
```

- ▶ Attention aux problèmes d'intégrité !

- ▶ Modification d'une table :

```
ALTER TABLE nom_table
```

```
MODIFY(nom_col1 TYPE1,nom_col2 TYPE2,...);
```

- ▶ Exemple : Un nom peut dépasser 20 caractères

```
ALTER TABLE Etudiant
```

```
MODIFY(Nom Char(25));
```

VI. Le langage de définition de données

2. Suppression et modification d'une table

- Changement de nom de tables ou de colonnes :

```
RENAME TABLE ancien_nom TO nouveau_nom ;
```

```
RENAME COLUMN nom_relation.ancien_nom_col TO  
nouveau_nom_col ;
```

- Exemple :

```
RENAME COLUMN Etudiant.DateNaissance  
TO BirthDay;
```