Université de Gabes
FACULTE DES SCIENCES DE GABES
Département Informatique

Fondements des bases de données

SECTION :LGLSI 1

AU. 2023-2024

Chapitre 5

Le Langage SQL

Introduction

- *SQL signifie « Structured Query Language », c'est à dire langage d'interrogation structuré*
- *SQL renferme quatre familles de commandes qui sont :*

  ☆ *Les commandes de définition de la structure de la base de données.*

  ☆ *Les commandes de la manipulation de données.*

  ☆ *Les commandes de recherche de données.*

  ☆ *Les commandes de modification de la structure de la BD de générer des pages web dynamiques ;*

Types des données

- Les types numériques : Nombres entiers ou décimaux.
- Les types BIT : permet de ranger une valeur booléenne.
- Les types chaîne de caractères : les constantes chaînes de caractères sont entourées par des apostrophes ('). Si la chaîne contient des apostrophes, celles-ci doivent être doublées (exemple : 'aujourd''hui').
- Les types temporels : pour les dates (DATE : réserve deux chiffres pour le mois et le jour et quatre pour l'année). Pour l'heure (TIME).

Types des données

- Exemples : les types de données les plus courants sont :

  - NUMBER (INT) : entier ou décimal de 40 positions maximum

  - NUMBER (taille_max), NUMBER (taille_max, décimale): taille_max <=105

  - CHAR (taille_max), VARCHAR (taille_max) : taille_max <=240

  - LONG, LONG VARCHAR : taille jusqu'à 65535

  - DECIMAL

Opérations

Langage de Définition des Données (LDD)

- Create
- Alter
- Drop

Langage de Manipulation des données (LMD)

- Insert
- Update
- Select

Langage d'interrogation de Données (LID)

- Select

Langage de Définition des Données (LDD)

1- Commande de création d'une Table

CREATE TABLE <nom_table> ({<defatt>}n) ;

- ***nom_table:*** sert à identifier un objet de type relation après sa création

- ***defatt*** permet de décrire la liste des attributs (ou constituants) de la relation

<nom_attribut> <type_attribut> [(<taille_attribut>)]
[NULL / NOT NULL]

- ***nom_attribut*** sert à identifier un constituant de la relation
- ***type_attribut*** sert à identifier le type de donnée qu'on peut associer à l'attribut
- ***taille_attribut*** est facultatif ? quand il est utilisé, il permet d'indiquer le nombre maximum de positions accepté par l'attribut
- ***NULL*** et ***NOT NULL*** sont à la fois facultatifs et exclusifs. Par défaut, c'est le paramètre NULL qui est prix en considération. Dans ce cas, l'attribut concerné par ce paramètre est autorisé à accepter des valeurs indéterminées. Par contre, si la clause NOT NULL est explicitement spécifiée, dans ce cas , on exige une valeur significative à chaque saisie d'une nouvelle valeur et à chaque modification d'une valeur existante.

Définition des contraintes d'intégrité

- ***PRIMARY KEY***, et aucune des colonnes qui composent cette clé ne doit avoir une valeur NULL

Constraint Nom_contrainte primary key (attribut_clé1,attribut_clé2,…)

- ***FOREIGN KEY***

Constraint Nom_contrainte foreign key (attribut_clé_étrangère) references Nom_table(attribut_clé_primaire)

PRIMARY KEY

- **Exemple 1 : Créer la table Produit en supposant que Numprod est sa clé primaire**

Solution 1 :

```
CREATE TABLE Produit
   (Numprod_number(6)  primary key,
   Desprod varchar(15),
   Couleur char,
   Poids number(8,3),
   Qte_stk number(7,3),
   Qte_seuil number(7,3),
   Prix number(10,3));
```

Solution 2:

```
CREATE TABLE Produit
      (Numprod_number(6),
      Desprod varchar(15),
      Couleur char,
      Poids number(8,3),
      Qte_stk number(7,3),
      Qte_seuil number(7,3),
      Prix number(10,3)
      Constraint Pk1 primary key (Numprod));
```

Dans le cas d'une clé primaire composée ,une seule solution possible

Cas d'une clé primaire composée

Exemple 2 : Créer la table

LigneCommande (NumCde, NumProd, QteCde)

Une seule solution possible

```
CREATE TABLE LigneCommande
      (NumCde number(8),
      NumProd number(6),
      QteCde number(7,3),
      constraint pk_LigneCde primary key ( NumCde, Numprod)) ;
```

FOREIGN KEY –Solution 1

- **Exemple : Créer la table produit sachant que la table magasin est déjà créée**

  MAGASIN (NumMag, Adresse, Surface)

  PRODUIT (Numprod, Desprod, Qte_stk, Prix, #CodMag)

Solution 1 : Clé étrangère comme contrainte de colonne

```
CREATE TABLE Produit
      (Numprod number(6) primary key ,
      Desprod varchar(15),
      Qte_stk number(7,3),
      Prix number(10,3),
      CodMag  number(6) references Magasin(NumMag));
```

FOREIGN KEY –Solution 2

- **Exemple : Créer la table produit sachant que la table magasin est déjà créée**

  MAGASIN (NumMag, Adresse, Surface)

  PRODUIT (Numprod, Desprod, Qte_stk, Prix, #CodMag)

Solution 1 : Clé étrangère comme contrainte de colonne

```
CREATE TABLE Produit
    (Numprod number(6) primary key ,
    Desprod varchar(15),
    Qte_stk number(7,3),
    Prix number(10,3),
    CodMag  number(6) references Magasin(NumMag));
```

FOREIGN KEY

- **Exemple : Créer la table produit sachant que la table magasin est déjà créée**

  MAGASIN (NumMag, Adresse, Surface)

  PRODUIT (Numprod, Desprod,, Qte_stk, Prix, #CodMag)

Solution 2 : Clé étrangère comme contrainte de table

```
CREATE TABLE Produit
    (Numprod number(6) primary key ,
    Desprod varchar(15),
    Qte_stk number(7,3),
    Prix number(10,3),
    CodMag  number(6),
    Constraint FK_Produit Foreign key(CodMAg) references Magasin(NumMag));
```

Définition des contraintes d'intégrité

- La contrainte de domaine ***CHECK(condition) :*** qui donne une condition que la ou les colonnes devront vérifier…

Constraint Nom_contrainte check (condition)

- La contrainte ***UNIQUE :*** interdit qu'une colonne contient deux valeurs identiques.

Définition des contraintes –Exemple 1

- Etant donné la relation Client suivante :

Client (Numcli, Nom, DateNaiss, NumEmploye#)

```
Create table Client (
    Numcli          number(3) NOT NULL,
    Nom             char(15),
    DateNaiss           date,
    NumEmploye      number(5) NOT NULL,
    Constraint pk_client primary key (Numcli),
    Constraint fk_employe foreign key (NumEmploye)
    references Employeur(NumEmploye),
    Constraint date_ok check(DateNaiss < sysdate));
```

Définition des contraintes –Exemple 2

- **Exemple 1 : Créer la table Produit en supposant que :**
  - **La saisie de la couleur du produit est obligatoire**
  - **Deux produits différents ne peuvent pas avoir la même désignation**

```
CREATE TABLE Produit
        (Numprod number(6) primary key ,
        Desprod varchar(15) unique,
        Couleur char not null,
        Poids number(8,3),
        Qte_stk number(7,3),
        Qte_seuil number(7,3),
        Prix number(10,3));
```

Définition des contraintes – Exemple 2

Exemple : Créer la table produit sachant que le poids doit être positif, la quantité en stock est comprise entre 0 et 1000 et est supérieure à la quantité seuil et que la couleur ne peut être que 'N', 'G' ou 'B'.

PRODUIT (Numprod, Desprod, Couleur, Poids, Qte_stk, Qte_seuil, Prix)

```
CREATE TABLE Produit
    (Numprod number(6) primary key ,
    Desprod varchar(15),
    Couleur char,
    Poids number(8,3),
    Qte_stk number(7,3),
    Qte_seuil number(7,3),
    Prix number(10,3),
    Constraint CK1_Produit check (Poids > 0),
    Constraint CK2_Produit check (Qte_stk >Qte_seuil ),
    Constraint CK3_Produit check (Qte_stk between 0 and 1000),
    Constraint CK4_Produit check (Couleur IN ('N','G','B') ));
```

2- Commande de modification de la structure d'une table

- Les cinq possibilités de modification de structure de table permettent :
  - d'ajouter des colonnes,
  - de modifier la structure d'une colonne,
  - de supprimer des colonnes existantes,
  - d'ajouter des contraintes,
  - de supprimer des contraintes.

2- Commande de modification de la structure d'une table

- **Ajout des colonnes :**

```
ALTER  TABLE  nom_table
        ADD  (nom-col1 type-col [(taille)] [DEFAULT valeur] [contrainte-col],
              nom-col2 type-col [(taille)] [DEFAULT valeur] [contrainte-col],
              nom-coln type-col [(taille)] [DEFAULT valeur] [contrainte-col] ) ;
```

- **Modification de la structure d'une colonne existante**

```
ALTER  TABLE  nom_table
      Modify  (nom-col1 type-col [(taille)] [DEFAULT valeur] [contrainte-col],
               nom-col2 type-col [(taille)] [DEFAULT valeur] [contrainte-col],
               nom-coln type-col [(taille)] [DEFAULT valeur] [contrainte-col] ) ;
```

- **Suppression de colonnes existantes**

```
ALTER  TABLE  nom_table
        Drop   (nom-col1, nom-col2, …, nom-coln) ;
```

2- Commande de modification de la structure d'une table-Exemple

Exemples :

MAGASIN (NumMag, Adresse, Surface)

PRODUIT (Numprod, Desprod, Couleur, Poids, Qte_stk, Qte_seuil, Prix, #CodMag)

Ajouter le champ Gérant de type caractère variable de taille 20 à la table magasin.

```
Alter table magasin
Add gerant varchar(20);
```

Ajouter le champ Ville à la table magasin de type caractère de taille 15 contenant par défaut Sfax.

```
Alter table magasin
Add ville char(15) default 'Sfax';
```

2- Commande de modification de la structure d'une table-Exemple

Exemples :

MAGASIN (NumMag, Adresse, Surface)

PRODUIT (Numprod, Desprod, Couleur, Poids, Qte_stk, Qte_seuil, Prix, #CodMag)

Modifier le champ Gérant de la table magasin de manière qu'il devienne caractère variable de taille 10.

```
Alter table magasin
Modify  gerant varchar(10);
```

Modifier la table Produit de manière à ce que la valeur par défaut de QteSeuil soit égale à 10.

```
Alter table produit
Modify  qteseuil default 10;
```

Supprimer les colonnes gérant et ville de Magasin

```
Alter table magasin
Drop (gerant,ville);
```

3- Commande de Suppression d'une table

```
DROP  TABLE  nom_table [ CASCADE CONSTRAINTS ] ;
```

- L'option CASCADE CONSTRAINTS permet de supprimer toutes les contraintes d'intégrité référentielles qui se reflètent aux clés primaires de la table à supprimer.

- Exemple : Supprimer dans l'ordre les tables Magasin et Produit

```
Drop  table  Magasin  CASCADE CONSTRAINTS ;

Drop  table  Produit;
```

3- Commande de Changement du nom d'une table

Rename ancien_nom **TO** nouveau_nom ;

- Exemple : modifier le nom de la table produit en Article

Rename Produit to article;

Langage de Manipulation des Données (LMD)

1- Commande d'insertion des données

INSERT INTO <nom_relation> [({<attribut>}n)]
VALUES ({<val>}n);

- ***nom_relation*** identifie la relation concernée par l'ajout de nouvelles données
- ***attribut*** répétitif et optionnel, permet de donner les attributs concernés par l'ajout de données. il devient obligatoire si l'ordre des valeurs données par le paramètre val ne respecte pas celui donné au moment de la création ou la modification de la structure de la relation. Ce paramètre est également obligatoire en cas où certains attributs de la relation ne sont pas concernés par l'ajout. Dans ce cas, le système affectera une valeur indéterminée (NULL) aux attributs de la relation qui ne seront pas spécifiés par le paramètre attribut.
- ***val*** répétitif, il permet de donner soit, de manière directe, soit de manière macro commandée la liste de valeurs qui doivent être insérées dans la relation.

1- Commande d'insertion des données -Exemple

```
INSERT  INTO  Client
        VALUES  (10, 'ISET', 'Gabes', 120000);

INSERT  INTO  Client (no_cli, raisoc_cli, ca_cli)
        VALUES  (20, 'ISG', 140000);
```

2- Commande de modification des données

UPDATE <relation>
SET ({<attribut> = <val>}n)
[WHERE <condition>] ;

- ***relation*** :se référer à la relation conservée par l'opérateur de mise à jour .

- Les paramètres ***attribut*** et ***val*** sont répétitifs. Le 1er sert à identifier un attribut de la relation dont le contenu doit être mis à jour. Le 2ème a pour objectif de donner la nouvelle valeur qui doit être effectuée à l'attribut.

- La clause ***WHERE*** est facultative. Si elle n'est pas spécifiée dans ce cas tous les n-uplets de la relation seront mis à jour. Par contre, si elle est présente, dans ce cas, seuls les n-uplets de la relation qui vérifient la condition donnée par le paramètre condition seront mis à jour.

2- Commande de modification des données -Exemple

- UPDATE Client
  SET raisoc_cli =' INSAT', adr_cli =' CITE OLYMPIQUE '
  WHERE no_cli = 10 ;

- UPDATE Client
  SET ca_cli = ca_cli + 10000;

2- Commande de modification des données -Exemple

Exemples :

Modifier la désignation du produit numéro 80 en Imprimante.

```
Update Produit
Set DesProd = 'Imprimante'
Where numProd = 80 ;
```

Majorer de 5% les prix des produits dont le prix est supérieur à 10.

```
Update Produit
Set Prix = Prix * 1.05
Where Prix > 10 ;
```

Modifier les quantités de tous les produits avec la valeur 10.

```
Update Produit
Set Qte_stk = 10;
```

3- Commande de suppression des données

DELETE FROM <relation>
[WHERE <condition>];

- **relation** :identifier la relation concernée par l'opération de suppression.

- La clause ***WHERE*** est facultative, si elle est absente, dans ce cas tous les n_uplets de la relation seront supprimés. Par contre, si elle est indiquée, dans ce cas seuls les n_uplets qui vérifient la condition de la clause WHERE seront supprimés.

3- Commande de suppression des données -Exemple

- **Delete From Client;**

- **Delete From Client**
  WHERE no_cli = 20 ;

3- Commande de suppression des données -Exemple

Exemples

Supprimer tous les Produits de couleur Blanche (‘B’)

```
Delete From Produit
Where couleur='B';
```

Supprimer toutes les lignes de la table Produit.

```
Delete From Produit ;
```

Langage d'interrogation de données (LID)

Commande de recherche de données

SELECT [distinct] {<attribut>}n / *
FROM {<relation> [,<variable>]}n
[WHERE <condition>]
[GROUP BY {<attribut>}n HAVING <codition>]
[ORDER BY {<attribut> [ASC/DESC]}n];

- **distinct :** exige la non duplication des résultats recherchés. Ce paramètre est facultatif.

Commande de recherche de données

- **distinct :** exige la non duplication des résultats recherchés. Ce paramètre est facultatif.

- ***attribut*** et * sont exclusifs, le 1er est répétitif : donne la liste des constituants appartenant à un ou plusieurs objets concernés par la recherche. Si tous les constituants sont demandés, le paramètre attribut est remplacé par *.

- ***relation*** et ***variable*** sont répétitifs. Le 1er se réfère à un objet de la base concerné par la sélection, le 2ième définit un nom de variable associé localement à l'objet dont le nom est donné juste avant. C'est ce nom de variable qui sert à enlever l'ambiguïté si elle existe.

- ***WHERE*** est facultative, définit la condition vérifiée par les données sélectionnées.

Remarques

- Opérateurs de la clause WHERE:

  IS NULL, IN (…),
  BETWEEN V1 AND V2,
  LIKE '<chaîne>' (avec % et _)

☆ **IS NULL** utilisé pour comparer des valeurs indéterminées.
☆ **IN** utilisé avec des ensembles de valeurs.
☆ **BETWEEN** utilisé avec les intervalles de valeurs.
☆ **LIKE** utilisé avec des chaînes de caractères, dans ces chaînes on peut utiliser les caractères joker % et _.

- Le paramètre condition de la clause WHERE peut être défini en utilisant la commande de recherche SELECT. On parle dans ce cas de SELECT imbriqué.

Exemples Recherche de données

☆ **Afficher tous les tuples d'une table**

```
Select  *  from  Client ;
```

☆ **Tri du résultat**

```
Select  *  from  Client
        Order by  Nom  DESC ;
```

☆ **Calculs :Calcul du prix tout taxe compris TTC**

```
Select  PrixUnit + PrixUnit * 0.18
        From  Produit ;
```

☆ **Projection : *sélectionner un ensemble de colonnes dans une table***

```
Select Nom, Prenom
        From Client ;
```

Exemples Recherche de données

☆ **Restriction : sélectionner les lignes satisfaisant à une condition logique effectuée sur leurs attributs.**

- **Afficher les clients qui habitent à tunis**

```
Select  *  from  Client
        Where  Ville = 'Tunis' ;
```

- **Commandes en quantité au moins égale à 3**

```
Select *  from  Commande
        Where Quantité >=3 ;
```

- **Produits dont le prix est compris entre 50 et 100.**

```
Select  *  from  Produit
        Where  PrixUnit between 50 and 100 ;
```

- **Commandes en quantité indéterminée.**

```
Select * from  Commande
        Where Quantité  is null ;
```

Exemples Recherche de données

- **les clients habitant une ville dont le nom se termine par unis.**

```
Select *  from Client
        Where  Ville like '%unis' ;
```

- **Prénom des clients dont les noms : Alimi, Zouari, Baccar**

```
Select  prénom  from  Client
        Where Nom in ('Alimi','Zouari','Baccar') ;
```

Remarque

Possibilité d'établir la négation pour tous ces prédicats

not between

not null

not like

not in

Commande de recherche de données

- ***GROUP BY*** est facultative, permet de répartir les tables en ensembles de lignes. Chaque élément de la liste des attributs de group by doit apparaître dans la liste de select. On ne peut pas créer des groupes qu'à partir d'objets sélectionnés.

- ***HAVING*** est une clause where pour groupes. De même que where limite les lignes, having limite les groupes, le plus souvent, on emploi having avec group by.

- ***ORDER BY*** est facultative, exige un classement (tri) des résultats recherchés. Ce classement se fait par rapport à un ensemble de constituants donnés par le paramètre attribut. Ces constituants doivent figurer parmi ceux qui sont donnés implicitement par le paramètre *.
  - Le classement : soit par ordre croissant (***ASC***), soit par ordre
  - décroissant (***DESC***), par défaut c'est l'ordre croissant qui
- sera pris en considération.

Commande de recherche de données-

Classification des résultats (Group BY)

nom	capitale	population	surface	sign	continent
Irlande	Dublin	3	70	IRL	Europe
Autriche	Vienne	8	83	A	Europe
Royaume-Uni	Londres	36	244	UK	Europe
Suisse	Berne	7	41	CH	Europe
USA	Washington	189	441	USA	Amerique

```
SELECT  continent, MIN(population), MAX(population), AVG(population),
SUM (surface), COUNT(*)

FROM  PAYS

GROUP BY continent  ;
```

continent	'min(pop)'	'max(pop)'	'avg(pop)'	'sum(surface)'	count
Amerique	189	189	189	441	1
Europe	3	36	13.5	438	4

Commande de recherche de données-

Classification des résultats (Group BY)

Table *Commande*		
noCommande	dateCommande	noClient
1	01/06/2000	10
3	02/06/2000	10
4	05/07/2000	10
2	02/06/2000	20
6	09/07/2000	20
5	09/07/2000	30
7	15/07/2000	40
8	15/07/2000	40

noClient	nombreCommandes
10	3
20	2
30	1
40	2

```
SELECT    noClient, count(*) nbreCommandes

FROM  commande

GROUP BY noClient  ;
```

Commande de recherche de données-Tri des résultats

- **Par défaut, le tri est croissant (asc)**
- **Exemples**

Donner la liste des produits ordonnés par ordre croissant de leurs prix.

```
SELECT *
FROM produit
ORDER BY  prix ;
```

Donner la liste des produits ordonnés par ordre croissant de leurs prix et décroissant de leurs désignations.

```
SELECT *
FROM produit
ORDER BY  prix, des_prod desc ;
```

Exercice

Numprod	Desprod	Couleur	Poids	Qte_stk	Qte_seuil	Prix
100	Ordinateur	R	15.2	30	2	100.275
800	disquette		20	24	4	125
200	Souris	G	30	12	5	200.450
102	Tapis	R	0.125	10	5	10

Afficher les numéros et désignations de tous les produits :

```
Select numprod, desprod
From Produit ;
```

Afficher les numéros et désignations des produits existants en stock avec une quantité > 20.

```
Select numprod,desprod
From Produit
Where qte_stk > 20 ;
```

Afficher les produits existants en stock avec une quantité > 20.

```
Select *
From Produit
Where qte_stk > 20 ;
```

Afficher les couleurs des différents produits.

```
Select Distinct couleur
From produit;
```

Afficher les numéros de produits dont la couleur n'a pas été saisie.

```
Select numprod
From produit
Where couleur is null;
```

Afficher les produits de couleur Rouge, Bleu ou Gris.

```
Select *
From Produit
Where couleur In ('R','B','G');
```

Afficher les numéros des produits dont le prix est compris entre 100 et 200.

```
Select numprod
From produit
Where prix between 100 and 200.
```

Afficher les produits dont la désignation commence par 'o'.

```
Select *
From produit
Where desprod like 'o%';
```

Afficher les numéros et désignations des produits dont les noms commencent par o ou par s

```
Select numprod,desprod
From produit
Where desprod like ‘o%' or desprod like 's%';
```

Afficher les désignations des produits contenant r en deuxième position de la designation et existant en stock avec une quantité > 20.

```
Select desprod
From produit
Where desprod like '-r%' and qte_stk > 20;
```

Afficher par couleur la quantité totale de produits.

```
Select couleur, sum(qte_stk) "Qte totale"
From produit
Group by couleur ;
```

Afficher par couleur la quantité totale d'ordinateurs.

```
Select couleur, sum(qte_stk) "Qte totale"
From produit
Where upper(desprod)='ORDINATEUR'
Group by couleur ;
```

Afficher par couleur et designation le nombre de produits.

```
Select desprod, couleur, count(*) "Nombre"
From produit
Group by desprod,couleur;
```

Afficher par couleur la quantité totale des produits, supérieure à 100.

```
Select couleur,sum(qte_stk) "Qte totale"
From produit
Group by couleur
Having sum(qte_stk)> 100;
```

Afficher par couleur la quantité totale des ordinateurs, supérieure à 100.

```
Select couleur,sum(qte_stk) "Qte totale"
From produit
Where upper (desprod) ='ORDINATEUR'
Group by couleur
Having sum(qte_stk)> 100;
```

Afficher les couleurs des produits ayant une quantité totale > 100

```
Select couleur
From produit
Group by couleur
Having sum(qte_stk)> 100;
```