UNIVERSITÉ POLYTECHNIQUE DE BINGERVILLE

BASE DE DONNEES

Chapitre 4.

Le langage SQL (Structured Query Language)

Partie 1

Dr. Coulibaly Tiékoura

Plan du Chapitre 4

I.
Introduction

I.Introduction

SQL: **Structured Query Language** est un langage Standard permettant à un client de communiquer des instructions à la base de données. Il se décline en quatre parties :

— **le DDL (Data definition language)** comporte les instructions qui permettent de définir la façon dont les données sont représentées.

— **le DML (Data manipulation language)** permet d'écrire dans la base et donc de modifier les données.

— **le DQL (Data query language)** est la partie la plus complexe du SQL, elle permet de lire les données dans la base à l'aide de requêtes.

— **le DCL (Data control language)**, qui permet de gérer les droits d'accès aux données.

A cela s'ajoute des extensions procédurales du SQL (appelé PL/SQL en Oracle). Celui-ci permet d'écrire des scripts exécutés par le serveur de base de données.

1.1. Objectifs de SQL

- Créer la structure de la base de données et de ses table
- Exécuter les tâches de base de la gestion des données,
- telle que l'insertion, la modification et la suppression de
- données des tables
- Effectuer des requêtes simples ou complexes

1.2. Principales instructions:

- Définitions (LDD)
  CREATE, DROP, ALTER
- Mises à jour (LMD)
  INSERT, UPDATE , DELETE
- Interrogations (LMD)
  SELECT
- Contrôle d'accés aux données
  GRANT, REVOKE
- Gestion de transactions
  COMMIT, ROLLBACK

1.3. Connexion à une base de données:

- Dans une base de données relationnelle, les données sont stockées dans des tables. Une table est un tableau à deux entrées. Nous allons nous connecter à une base de données pour observer les tables.

- La méthode la plus simple pour s'initier à **mysql** est d'utiliser un kit de XAMPP, WAMP, etc. Vous disposez dans ce cas d'une option vous permettant d'ouvrir une **console mysql**.

- **La liste des bases de données** stockées dans le serveur s'obtient avec l'instruction:

  show databases

- **On se connecte à l'une des bases de données** avec l'instruction:

  use nomdelabase

1.4. Consultation des tables:

- Une fois dans la base, on obtient **la liste des tables** avec l'instruction:

  show tables

- On affiche la **liste des colonnes d'une table (ou structure d'une table)** avec l'instruction:

  desc PRODUIT ou ***describe PRODUIT***

- **Le contenu d'une table** s'affiche avec l'instruction:

  SELECT *
  FROM PRODUIT

1.5. Organisation relationnelle des données:

Nous utiliserons pour commencer les types suivants :

— numérique entier : int
— numérique à point fixe : number (Oracle seulement)
— numérique à point flottant : real
— chaîne de caractères : varchar(taille) ou varchar2(taille) (Oracle seulement).

❑Créer une base de données

CREATE DATABASE *maBase*

ou

CREATE DATABASE IF NOT EXISTS *maBase*

NB: l'option « IF NOT EXISTS permet de ne pas retourner d'erreur si une base du même nom existe déjà. La BD ne sera pas écrasée.

❑Créer des tables

Exemple de création de table :

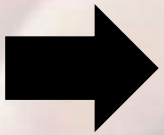
```
CREATE TABLE CLIENT(
        numcli int,
        nomcli varchar(32));
desc CLIENT;

Field   Type      Null    Key     Default  Extra
numcli  int       YES             NULL
nomcli  varchar(32)       YES             NULL
```

1.5. Organisation relationnelle des données:

❑Ajouter une ligne dans une table:

Exemple d'insertion de données dans une table :

```
INSERT INTO CLIENT (numcli, nomcli)
VALUES (1, 'Marcel'), (2, 'Gégé');
SELECT * FROM CLIENT;
```



numcli	nomcli
1	Marcel
2	Gégé

Attention, chaque commande SQL se termine par un point-virgule !

❑Suppression d'une table:

Une table se supprime avec l'instruction DROP TABLE.

DROP TABLE CLIENT ;

1.5. Organisation relationnelle des données:

Utilisez l'instruction **ALTER TABLE** pour,

- Ajouter une colonne
- Modifier une colonne existante
- Supprimer une colonne
- Ajouter une contrainte
- Supprimer une contrainte

❑ **Pour ajouter, modifier ou supprimer des colonnes**.

a) Ajouter une colonne avec la clause ADD:

```
ALTER TABLE department
ADD job VARCHAR(9);
```

TAF: Ajouter la colonne telephone à la table Client créée.

1.5. Organisation relationnelle des données:

b) Modifier une colonne au niveau du type de données (MODIFY):

```
ALTER TABLE etudiant
MODIFY last_name VARCHAR(30);
```

c) Renommer une colonne:

```
ALTER TABLE nom_table
CHANGE colonne_ancien_nom colonne_nouveau_nom type_donnees
```

d) Renommer une colonne:

```
ALTER TABLE nom_table
DROP nom_colonne
```

ou bien

```
ALTER TABLE nom_table
DROP COLUMN nom_colonne
```

Suppression

L'expression

```
DELETE FROM <NOMTABLE>
WHERE <CONDITION>
```

❑ Efface de la table NOMTABLE toutes les lignes vérifiant condition. Attention ! La commande

```
DELETE FROM <NOMTABLE>
```

efface toutes les lignes de la table NOMTABLE !

Mise à jour (1/2)

L'expression

```
UPDATE <NOMTABLE>
SET <colonne_1> = <valeur_1>,
<colonne_2> = <valeur_2>,
...,
<colonne_n> = <valeur_n>
WHERE <CONDITION>
```

modifie les lignes de la table NOMTABLE vérifiant condition. Elle affecte au champ colonnei la valeur valeuri.

Mise à jour (2/2)

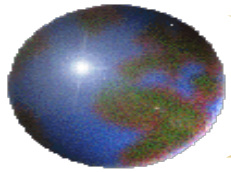
Par exemple,

```
UPDATE CLIENT
SET prenomcli = 'Dark'
WHERE nomcli = 'Vador'
```

affecte la valeur 'Dark' aux champs *prenomcli* de toutes les lignes dont la valeur *nomcli* est égale à 'Vador'. Il est possible, dans une modification, d'utiliser les valeurs des autres champs de la ligne, voire même l'ancienne valeur de ce champ. Par exemple,

```
UPDATE OPERATION
SET montantoper = montantoper + 5000
```

augmente les montants de toutes les opérations bancaires de 5000 (choisissez l'unité !).



II.
Contraintes déclaratives

II. Contraintes déclaratives

❑Valeurs par défaut:

```
create table client
        (
        numcli int,
        nom varchar(256) default 'Moi',
        prenom varchar(256)
        )
```

fait de 'Moi' le nom par défaut.

❑ Champs non renseignés:

```
create table client
        (
        numcli int,
        nom varchar(256) NOT NULL,
        prenom varchar(256) NOT NULL
        )
```

force la saisie des champs nom et prénom.

II. Contraintes déclaratives

❑Clé primaire:

Une clé primaire est :

— toujours renseignée

— unique

On peut préciser **PRIMARY KEY** dans la création de table

```
create table client
        (
        numcli int PRIMARY KEY,
        nom varchar(256),
        prenom varchar(256)
        )
```

La colonne *numcli* est clé primaire, toute insertion ne respectant pas la contrainte de clé primaire sera refusée par le SGBD.

II. Contraintes déclaratives

❑Clé étrangère:

Dans le cas où l'on souhaite garder en mémoire des factures émises par des clients, la façon de faire est de créer une deuxième table contenant la liste des factures :

```
create table facture
        (
        numfact int PRIMARY KEY,
        montantFacture int,
        numcli int REFERENCES CLIENT(numCli)
        );
```

Le champ **numCli** dans cette table est **clé étrangère**, ce qui signifie qu'une ligne ne pourra être insérée dans la table facture que si le *numcli* de cette ligne existe dans la colonne *numcli* de la table client.

La syntaxe est: **REFERENCES <nomtable> (<nomcolonne>)**

❑Syntaxe alternative:

Il est possible de définir les contraintes après la création d'une table.

```
ALTER TABLE nomtable
ADD [CONSTRAINT nomcontrainte] descriptioncontrainte;
```

II. Contraintes déclaratives

❑Syntaxe alternative:

- *Descriptioncontrainte d'une clé primaire* :

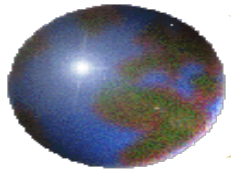
```
PRIMARY KEY( colonne1 , ... , colonnen )
```

- *Descriptioncontrainte d'une clé étrangère*:

```
FOREIGN KEY( colonne1 , ... , colonnen )
REFERENCES tablereferencee (colonne1 , ... , colonnen )
```

❑Il est aussi possible de placer une *descriptioncontrainte* dans le **CREATE TABLE**. Par exemple,

```
create table facture
        (
        numfact int,
        montantFacture int,
        numcli int,
       PRIMARY KEY (numfact),
       FOREIGN KEY nucli REFERENCES CLIENT(numcli)
       );
```



II. Contraintes déclaratives

❑Syntaxe alternative:

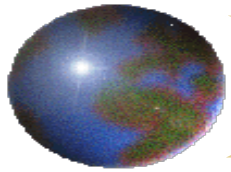
On remarque qu'il est possible de nommer une contrainte. C'est utile si on souhaite la supprimer : `ALTER TABLE nomtable DROP CONSTRAINT nomcontrainte;`

- ❑ Pour **lister les contraintes sous Oracle**, on utilise la commande :

```
SELECT * FROM USER_CONSTRAINTS;
```

- ❑ Pour **lister les contraintes sous MySQL**, on utilise la commande :

```
SHOW TABLE STATUS;
```



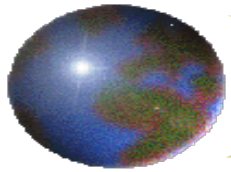
II. Contraintes déclaratives

❑EXERCICE D'APPLICATION 1:

1) A partir du modèle relationnel ci-dessous, construire le MCD équivalent à l'aide d'un logiciel de conception de votre choix.

Client(*numcli, nomcli, prenomcli, adressecli, contact, dateNaissance)*
Facture(*numfact, montantfact, #numcli)*
Produit(*numprod, libellé, prix, #numcli, #idEntr)*
Entreprise(*idEntr, nomEntr, ville)*
Employé(*matricule, nomEmpl, prenomEmpl, salaire, commission, email, âge)*
Travailler(*#idEntr, #matricule, dateEmbauche)*

1) Créer en code SQL à partir du SGBD MyQL la base de données correspondante à ce système d'information. On la nommera **ma_base.**
2) Créer les différentes tables en prenant le soins d'ajouter les différentes contraintes, les valeurs par défaut, la saisie obligatoire de certains champs.
3) Alimenter chaque table de votre BD par au moins 15 occurrences.



III.
Sélection de lignes et colonnes: Instructions SQL de base

3.1. Format des requêtes SQL et Syntaxe du select

- **SELECT:** spécifie les colonnes qui doivent apparaître dans les résultats
- **FROM:** spécifie la table ou les tables à utiliser
- **WHERE:** filtre les lignes selon une condition donnée
- **GROUP BY:** forme des groupes de lignes de même valeur de colonne
- **HAVING:** filtre les groupes sujets à une certaine condition
- **ORDER BY:** spécifie l'ordre d'apparition des données dans le résultat

3.1. Format des requêtes SQL et Syntaxe du select

❑ Syntaxe du selectSQL

```
SELECT      {[ALL|DISTINCT] expression [AS nomColonne]
                    [,expression [AS nomColonne]]…}|*
FROM        table [AS nomTable [(nomColonne[,nomColonne])]]
           [,table [AS nomTable [(nomColonne[,nomColonne])]]]…
[WHERE      conditionSQL]
[GROUP BY   nomColonne [,nomColonne]…
[HAVING     conditionSQL]
[ORDER BY   nomColonne [ASC|DESC] [,nomColonne[ASC|DESC]]…]
```

❑ Instruction SELECT de base:

```
SELECT *|{[DISTINCT] column|expression [alias],...}
FROM   table;
```

3.2. Projection d'une table: Sélection de toutes les colonnes

❑ **Sélectionner tout les départements**

```
SELECT *
FROM departments;
```

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting		1700

TAF: Afficher tous les produits de la tables Produit de votre base de données « ma_base »

3.3. La clause DISTINCT

- **Produire les *noClient* et *dateCommande* de toutes les Commandes**

```
SELECT noClient, dateCommande
FROM Commande

SELECT ALL noClient, dateCommande
FROM Commande

SELECT DISTINCT noClient, dateCommande
FROM Commande
```

$$\pi_{noClient,\ dateCommande}(Commande)$$

noClient	dateCommande
10	01/06/2000
20	02/06/2000
10	02/06/2000
10	05/07/2000
30	09/07/2000
20	09/07/2000
40	15/07/2000
40	15/07/2000

Redondance!

noClient	dateCommande
10	01/06/2000
20	02/06/2000
10	02/06/2000
10	05/07/2000
30	09/07/2000
20	09/07/2000
40	15/07/2000

TAF: Afficher les *numcli*, *adressecli* et contacts de tous les clients de la tables client de votre base de données « ma_base »

3.4. Utiliser des opérateurs arithmétiques

```
SELECT last_name, salary, salary + 300
FROM   employees;
```

LAST_NAME	SALARY	SALARY+300
King	24000	24300
Kochhar	17000	17300
De Haan	17000	17300
Hunold	9000	9300
Ernst	6000	6300

TAF: Afficher les *noms et prénoms des employés de votre BD « ma_base »*, leur salaire mensuel ainsi que leur salaire annuel augmenté de 10 000 F.

3.5. Utiliser des Alias de colonne

```
SELECT last_name AS name, commission_pct comm
FROM   employees;
```

```
SELECT last_name "Name" , salary*12 "Annual Salary"
FROM   employees;
```

Name	Annual Salary
King	288000
Kochhar	204000
De Haan	204000

TAF: Reprendre l'exercice précédant en renommant les attributs salaire en « ***salaire_mensuel*** » et le salaire annuel augmenté de 10000 en « ***sal_annuel_bonus*** »

CONCAT(colonne1, colonne2)

3.6. Utiliser l'opérateur de concaténation (CONCAT)

- Lie des colonnes ou des chaînes de caractères à d'autres colonnes.
- Est représenté par deux barres verticales (||).
- Crée une colonne résultante qui est une expression de type caractère.

```
SELECT  CONCAT (last_name, job_id) AS « Employees »
FROM employees;
```

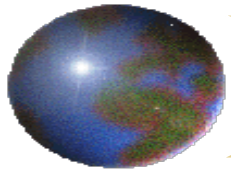
Employees
KingAD_PRES
KochharAD_VP
De HaanAD_VP

```
SELECT  CONCAT (last_name,  '' is a '' job_id) AS
« Employee Details »
FROM employees;
```

Employee Details
King is a AD_PRES
Kochhar is a AD_VP
De Haan is a AD_VP
Hunold is a IT_PROG
Ernst is a IT_PROG
Lorentz is a IT_PROG

TAF: En vous inspirant de l'exemple précédent, représentez dans une même colonne, les occurrences de la table Produit comme suit: (libellé du produit) **a pour prix** (le prix du produit) et renommez ladite colonne: « **coût du produit** »

IV.
Utilisation des conditions



IV. Utilisation des conditions

4.1. La clause WHERE

- **Objectifs**: Limiter et trier les lignes extraites par une interrogation.
- La clause **WHERE** suit la clause **FROM**
- **Syntaxe**:

```
SELECT *|{[DISTINCT] column|expression [alias],...}
FROM   table
[WHERE condition(s)];
```

- **Exemple**: - Extraire tous les étudiants de la filière MIAGE
  - Extraire tous les employés de plus de 20 ans d'ancienneté
  - etc.

4.2. Opérateurs de comparaison

Operator	Meaning
=	Egal à
>	Supérieur à
>=	Supérieur ou égal à
<	Inférieur à
<=	Inférieur ou égal à
<>	Non égal à
BETWEEN ...AND...	Entre deux valeurs (incluses)
IN(set)	Correspond à une valeur quelconque d'une liste
LIKE	Correspond à un modèle de caractère
IS NULL	Est une valeur NULL

IV. Utilisation des conditions

4.3. Opérateur de comparaison (=/ >/<)

```
SELECT employee_id, last_name, job_id, department_id
FROM   employees
WHERE  department_id = 90;
```

1

EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
100	King	AD_PRES	90
101	Kochhar	AD_VP	90
102	De Haan	AD_VP	90

```
SELECT last_name, job_id, department_id
FROM   employees
WHERE  last_name = 'Whalen';
```

2

- Les chaînes de caractères et les dates sont incluses entre apostrophes.
- Les valeurs de type caractère distinguent les majuscules des minuscules et les valeurs de date sont sensibles au format.
- Le format de date par défaut est **DD-MM-RR**

3

```
SELECT last_name, salary
FROM   employees
WHERE  salary <= 3000;
```

LAST_NAME	SALARY
Matos	2600
Vargas	2500

TAF: Utilisez la BD de référence « ma_base » pour répondre aux requêtes suivantes:

1) sélectionnez les employés qui ont atteint l'âge de la retraite (60 ans)
2) sélectionnez les clients qui habite la commune de yopougon

IV. Utilisation des conditions

4.4. Condition BETWEEN

- La condition **BETWEEN** permet d'afficher les lignes en fonction d'une plage de valeurs.

```
SELECT last_name, salary
FROM   employees
WHERE  salary BETWEEN 2500 AND 3500 ;
```

Limite inférieure — Limite supérieure

LAST_NAME	SALARY
Raja	3500
Davies	3100
Matos	2600
Vargas	2500

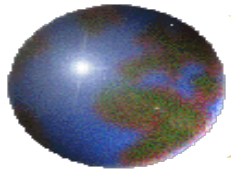
4.5. Condition IN

- On utilise la condition d'appartenance **IN** afin de tester les valeurs d'une liste.

```
SELECT employee_id, last_name, salary, manager_id
FROM   employees
WHERE  manager_id IN (100, 101, 201) ;
```

EMPLOYEE_ID	LAST_NAME	SALARY	MANAGER_ID
202	Fay	6000	201
200	Whalen	4400	101
205	Higgins	12000	101
101	Kochhar	17000	100
102	De Haan	17000	100

TAF: Appliquez les opérateurs BETWEEN et IN aux tables de votre choix de la BD de référence « ma_base »



IV. Utilisation des conditions

4.6. Condition LIKE

▫ La condition LIKE permet d'effectuer des recherches de chaînes de caractères valides via l'utilisation de caractères génériques.

 Les conditions de recherche peuvent contenir soit des caractères littéraux, soit des nombres :

- LIKE '%a' : le caractère **"%"** est un caractère joker qui remplace tous les autres caractères. Ainsi, ce modèle permet de rechercher toutes les chaines de caractère **qui se termine par un "a".**
- LIKE 'a%' : permet de rechercher toutes les lignes de "colonne" **qui commence par un "a".**
- LIKE '%a%' : est utilisé pour rechercher tous les enregistrements **qui utilisent le caractère "a".**
- LIKE 'pa%on' : ce modèle permet de rechercher les chaines qui commence par "pa" et qui se terminent par "on", comme "pantalon" ou "pardon".
- LIKE 'a_c' : le caractère "_" (underscore) peut être remplacé par n'importe quel caractère, mais un seul caractère uniquement (alors que le symbole pourcentage "%" peut être remplacé par un nombre incalculable de caractères . Ainsi, ce modèle permet de retourner les lignes "aac", "abc" ...ou "azc".

4.6. Condition LIKE

```
SELECT first_name
FROM   employees
WHERE first_name LIKE 'S%' ;
```

```
SELECT last_name
FROM   employees
WHERE  last_name LIKE '_o%';
```

LAST_NAME
Kochhar
Lorentz
Mourgos

4.7. Condition NULL

- Elle permet de tester la présence de valeurs NULL

```
SELECT last_name, manager_id
FROM   employees
WHERE  manager_id IS NULL;
```

LAST_NAME	MANAGER_ID
King	

TAF: Appliquez les opérateurs **LIKE** et **NULL** aux tables de votre choix de la BD de référence « ma_base »

4.8. Double conditions avec les opérateurs AND, OR, NOT

Opérateur	Signification
AND	Renvoie TRUE si les deux conditions sont vraies.
OR	Renvoie TRUE si l'une des deux conditions est vraie.
NOT	Renvoie TRUE si la condition qui suit est fausse.

a) L'opérateur AND

L'opérateur AND nécessite que les deux conditions soient vraies

- **Sélectionner les *Articles* dont le *prix est inférieur à $20.00 et le numéro est supérieur à 30***

```
SELECT *
FROM Article
WHERE prixUnitaire < 20 AND noArticle > 30
```

noArticle	description	prixUnitaire
60	Erable argenté	15.99
70	Herbe à puce	10.99
95	Génévrier	15.99

$$\sigma_{prixUnitaire < 20.00 \text{ ET } noArtide > 30}(Article)$$

4.8. Double conditions avec les opérateurs AND, OR, NOT

a) L'opérateur AND (suite)

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary >=10000
AND    job_id LIKE '%MAN%';
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
149	Zlotkey	SA_MAN	10500
201	Hartstein	MK_MAN	13000

b) L'opérateur OR

L'opérateur OR nécessite que l'une des deux conditions soient vraies

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary >=10000
OR     job_id LIKE '%MAN%';
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
100	King	AD_PRES	24000
101	Kochhar	AD_VP	17000
102	De Haan	AD_VP	17000
124	Mourgos	ST_MAN	5800
149	Zlotkey	SA_MAN	10500
174	Abel	SA_REP	11000

4.8. Double conditions avec les opérateurs AND, OR, NOT

c) L'opérateur NOT

```
SELECT last_name, job_id
FROM   employees
WHERE  job_id
       NOT IN ('IT_PROG', 'ST_CLERK', 'SA_REP') ;
```

LAST_NAME	JOB_ID
King	AD_PRES
Kochhar	AD_VP
De Haan	AD_VP
Mourgos	ST_MAN
Zlotkey	SA_MAN
Whalen	AD_ASST

TAF: Utilisez la BD de référence « ma_base » pour répondre aux requêtes suivantes:

1) sélectionnez les employés qui ont atteint l'âge de 45 ans et dont le salaire est inférieur à 250 000 F
2) sélectionnez les produits qui sont moins chère (prix inférieur à 10 000) ou qui sont fabriqués par l'entreprise OLAM (d'identifiant E025)
3) Donner la liste des clients qui n'habitent pas les communes d'Abobo, de yopougon et de Koumassi.

IV. Utilisation des conditions

4.9. La clause ORDER BY

 La clause **ORDER BY** peut être utilisée avec l'instruction SELECT pour trier les données de champs spécifiques de **manière ordonnée**. Il est utilisé pour trier le jeu de résultats par ordre croissant ou décroissant.

- ✓ ASC : ordre croissant (par défaut)
- ✓ DESC : ordre décroissant

 La clause ORDER BY vient en dernier dans l'instruction SELECT:

```
SELECT   last_name, job_id, department_id, hire_date
FROM     employees
ORDER BY hire_date DESC ;
```

LAST_NAME	JOB_ID	DEPARTMENT_ID	HIRE_DATE
King	AD_PRES	90	17-JUN-87
Whalen	AD_ASST	10	17-SEP-87
Kochhar	AD_VP	90	21-SEP-89

```
SELECT employee_id, last_name, salary*12 annsal
FROM   employees
ORDER BY annsal ;
```

TAF: 1) Donnez les matricules des employés, les identifiants respectifs des entreprises dans lesquelles ils travaillent et les dates d'embauche, par ordre d'ancienneté selon la date d'embauche.
2) Classez les employés selon leur salaire annuel décroissant.