



# Chapitre 5

# Langage SQL

**Enseignante: Nour H. BEN SLIMEN ATTAWI**

# Introduction

- ▶ Une **base de données** est un ensemble d'informations structurées.
- ▶ Un **SGBDR** (Système de Gestion de Bases de Données Relationnel) est un logiciel qui permet de :
  - stocker,
  - consulter,
  - modifier,
  - supprimerles données de la base de données.
- ▶ Un **SGBDR** stocke les informations dans des tables.

# Introduction

- ▶ **SQL (Structured Query Language)** introduit par IBM et commercialisé tout d'abord par ORACLE
- ▶ SQL est devenu le langage standard pour décrire et manipuler les BDR
- ▶ Les commandes SQL :
  - ▶ **Langage de Définition des Données (LDD):** Ce langage permet la définition et la mise à jour de la structure de la base de données (tables, attributs, ...).
  - ▶ **Langage de Manipulation des Données (LMD):** Ce langage permet de manipuler les données de la base et de les mettre à jour.
  - ▶ **Le Langage d'Interrogation de Données (LID) :** Ce langage permet de rechercher des informations utiles en interrogeant la base de données. Certains considèrent ce langage comme étant une partie du LMD.

# LES COMMANDES 1 / 2

- ▶ Langage de Définition des Données (LDD):
  - ▶ **CREATE**: Création de tables
  - ▶ **ALTER**: Modification de tables
  - ▶ **DROP**: Suppression de tables
- ▶ Langage de Manipulation des Données (LMD) :
  - ▶ **INSERT**: Insertion de lignes dans une table
  - ▶ **UPDATE**: Mise à jour de lignes dans une table
  - ▶ **DELETE**: Suppression de lignes dans une table
- ▶ Langage d'Interrogation des Données (LID) :
  - ▶ **SELECT**: Interrogations diverses

# LANGAGE DE DÉFINITION DE DONNÉES (LDD)

# LANGAGE DE DÉFINITION DE DONNÉES (LDD)

Les commandes de définition des données en SQL sont:

- ▶ CREATE TABLE
- ▶ DROP TABLE
- ▶ ALTER TABLE

# Créer une table

- ▶ SQL permet de créer des relations sous forme de tables et de définir lors de la création des contraintes d'intégrité variées sur les attributs. Une commande de création permet donc de préciser le nom de la table et de définir les éléments de la table correspondant aux colonnes ou aux contraintes.

- ▶ **Syntaxe:**

```
CREATE TABLE table_name (  
    column_1 data_type column_constraint,  
    column_2 data_type column_constraint,  
    ...  
);
```

- ▶ **Exemple:**

```
CREATE TABLE Agence (  
    NumAg number (4) PRIMARY KEY,  
    NomAg varchar2(10),  
    VilleAG varchar2(20)  
);
```

# Supprimer une table

- ▶ La commande **DROP TABLE** en SQL permet de supprimer définitivement une table d'une base de données.

## Exemple:

- ▶ Créer la table 'persons':

```
CREATE TABLE persons (  
    person_id NUMBER,  
    first_name VARCHAR2(50) NOT NULL,  
    last_name VARCHAR2(50) NOT NULL,  
    PRIMARY KEY(person_id)  
);
```

- ▶ Supprimer la table 'persons':

```
DROP TABLE persons;
```



# Modifier un objet dans une table

## ALTER TABLE

Vous pouvez être amené à modifier à n'importe quel moment la structure d'une table créée dans la base de données. Les raisons classiques de modification d'une table sont les suivantes :

- Ajouter une colonne
- Supprimer une colonne
- Changer un nom de colonne
- Changer les types de données d'une colonne

### Syntaxe:

```
ALTER TABLE table_name action;
```

# Modifier un objet dans une table

## ALTER TABLE ADD

Pour ajouter une nouvelle colonne à une table, vous utilisez la syntaxe suivante :

### Syntaxe:

```
ALTER TABLE table_name  
ADD column_name type constraint;
```

### Exemple:

Par exemple, l'instruction suivante ajoute une nouvelle colonne nommée 'birthdate' à la table 'persons':

```
ALTER TABLE persons  
ADD birthdate DATE NOT NULL;
```


# Modifier un objet dans une table

## ALTER TABLE ADD

Pour ajouter plusieurs colonnes à une table en même temps, placez les nouvelles colonnes entre parenthèses comme suit :

```
ALTER TABLE table_name
ADD (
    column_name type constraint,
    column_name type constraint,
    ...
);
```

Dans cet exemple, l'instruction a ajouté deux nouvelles colonnes nommées 'phone' et 'email' à la table 'persons'.



```
ALTER TABLE persons
ADD (
    phone VARCHAR2(20),
    email VARCHAR2(100)
);
```

# Modifier un objet dans une table

## ALTER TABLE MODIFY


Pour modifier les attributs d'une colonne, vous utilisez la syntaxe suivante :

```
ALTER TABLE table_name  
  MODIFY column_name type constraint;
```

- Par exemple, l'instruction suivante modifie la colonne de 'birthdate' en une colonne à capacité NULL :

```
ALTER TABLE persons MODIFY birthdate DATE NULL;
```

Pour modifier plusieurs colonnes, vous utilisez la syntaxe suivante :



```
ALTER TABLE table_name  
  MODIFY ( column_1 type constraint,  
            column_1 type constraint,  
            ...);
```

# Modifier un objet dans une table

## ALTER TABLE MODIFY

- ▶ Par exemple, l'instruction suivante modifie la colonne 'phone' et 'email' en colonnes NOT NULL et étend la longueur de la colonne 'email' à 255 caractères :

```
ALTER TABLE persons  
    MODIFY(phone VARCHAR2(20) NOT NULL,  
           email VARCHAR2(255) NOT NULL);
```

# Modifier un objet dans une table

## ALTER TABLE DROP COLUMN

Pour supprimer une colonne existante d'une table, vous utilisez la syntaxe suivante :

```
ALTER TABLE table_name  
    DROP COLUMN column_name;
```

Cela permet de supprimer la colonne de la table ainsi que les données stockées dans cette colonne.

- L'exemple suivant supprime la colonne 'birthdate' de la table **persons** :

```
ALTER TABLE persons  
    DROP COLUMN birthdate;
```

# Modifier un objet dans une table

## ALTER TABLE DROP column

Pour supprimer plusieurs colonnes dans une table, utilisez la syntaxe suivante:



```
ALTER TABLE table_name  
    DROP (column_1,column_2,...);
```

- Par exemple, si on veut supprimer les colonnes 'phone ' et 'email' de la table 'persons':

```
ALTER TABLE persons  
    DROP ( email, phone );
```

# Modifier un objet dans une table

## ALTER TABLE RENAME column

Pour changer le nom d'une colonne dans une table, vous utilisez la syntaxe suivante :



```
ALTER TABLE table_name  
    RENAME COLUMN column_name TO new_name;
```

- Par exemple, changez le nom de la colonne 'first\_name' en 'forename' dans la table 'persons'

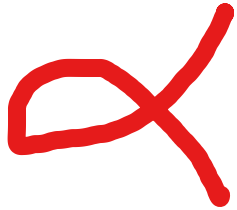
```
ALTER TABLE persons  
    RENAME COLUMN first_name TO forename;
```



# Modifier un objet dans une table

## ALTER TABLE RENAME table

- Pour donner un nouveau nom à une table, vous utilisez la syntaxe suivante :



```
ALTER TABLE table_name  
    RENAME TO new_table_name;
```

- Par exemple, si on veut changer le nom de la table 'persons' en 'people' :

```
ALTER TABLE persons RENAME TO people;
```

# LES CONTRAINTES EN SQL

Les contraintes sont les règles appliquées aux colonnes de données d'une table. Celles-ci sont utilisées pour limiter le type de données pouvant aller dans une table. Cela garantit l'exactitude et la fiabilité des données de la base de données.

Les contraintes peuvent être au niveau de la colonne ou de la table. Les contraintes de niveau de colonne ne sont appliquées qu'à une seule colonne, alors que les contraintes de niveau de table s'appliquent à l'ensemble de la table. Les contraintes les plus communes sont :

- ▶ NOT NULL
- ▶ DEFAULT
- ▶ UNIQUE
- ▶ CHECK
- ▶ PRIMARY KEY
- ▶ FOREIGN Key

# LES CONTRAINTES EN SQL

## Contrainte NOT NULL (1/2)

- Par défaut, une colonne peut contenir des valeurs **NULL**. Si vous ne souhaitez pas qu'une colonne ait une valeur **NULL**, vous devez définir une telle contrainte sur cette colonne en spécifiant que **NULL** n'est plus autorisé pour cette colonne.

### Syntaxe:

```
CREATE TABLE table_name (  
    ...  
    column_name data_type NOT NULL,  
    ...  
);
```

Il est possible d'ajouter une contrainte NOT NULL à une table existante en utilisant l'instruction ALTER TABLE.

```
ALTER TABLE table_name MODIFY (column_name NOT NULL);
```

# LES CONTRAINTES EN SQL

## Contrainte NOT NULL (2/2)

### Exemple :

Par exemple, la requête SQL suivante crée une nouvelle table appelée **Employés** et ajoute quatre colonnes (**Id**, **Nom**, **Age** et **Salaire**), dont **Id**, **Nom** et **Age** n'acceptent pas les valeurs **NULL**.

```
1 CREATE TABLE Employés(  
2     Id NUMBER      PRIMARY KEY  NOT NULL,  
3     Nom VARCHAR2 (20)      NOT NULL,  
4     Age  INT          NOT NULL,  
5     Salaire  DECIMAL (18, 2),  
6     );  
7
```


# LES CONTRAINTES EN SQL

## Contrainte DEFAULT

- La contrainte **DEFAULT** fournit une valeur par défaut à une colonne.

### Exemple :

Par exemple, le code SQL suivant crée la même table **Employés**, mais ici, la colonne Salaire est définie sur 5000.00 par défaut.



```
1 CREATE TABLE Employés(  
2     Id NUMBER      PRIMARY KEY  NOT NULL,  
3     Nom VARCHAR2 (20)      NOT NULL,  
4     Age  NUMBER              NOT NULL,  
5     Salaire  DECIMAL (18, 2) DEFAULT 5000.00,  
6 );  
7
```

# LES CONTRAINTES EN SQL

## Contrainte UNIQUE (1/2)

- ▶ La contrainte **UNIQUE** empêche que deux enregistrements aient des valeurs identiques dans une colonne
- ▶ Syntaxe:


```
CREATE TABLE table_name (  
    ...  
    column_name data_type UNIQUE  
    ...  
);
```

# LES CONTRAINTES EN SQL

## Contrainte UNIQUE (2/2)

### ► Exemple :

Par exemple, la requête SQL suivante crée la même table **Employés**, mais dans ce cas, la colonne **Nom** est définie sur **UNIQUE**, de sorte que vous ne pouvez pas avoir deux enregistrements portant le même **Nom**.



```
1 CREATE TABLE Employes(  
2     Id NUMBER          NOT NULL,  
3     Nom VARCHAR2 (20)  NOT NULL UNIQUE,  
4     Age  NUMBER          NOT NULL,  
5     Salaire  DECIMAL (18, 2),  
6 );  
7
```

# LES CONTRAINTES EN SQL

## Contrainte CHECK (1/2)

- La contrainte **CHECK** active une condition permettant de vérifier la valeur saisie dans un enregistrement. Si la condition est évaluée à false, l'enregistrement viole la contrainte et n'est pas entré dans la table.

### Syntaxe:

```
CREATE TABLE table_name (  
    ...  
    column_name data_type CHECK (expression),  
    ...  
);
```




# LES CONTRAINTES EN SQL

## Contrainte CHECK (2/2)

### Exemple :

Par exemple, la requête SQL suivante crée la même table **Employés**, mais dans ce cas, la colonne **Age** est définie sur **CHECK**, de sorte que vous ne pouvez pas avoir un employé de moins de 18 ans.



```
1 CREATE TABLE Employes(  
2     Id NUMBER          NOT NULL,  
3     Nom VARCHAR2 (20)   NOT NULL,  
4     Age  NUMBER          NOT NULL CHECK (Age >= 18),  
5     Salaire  DECIMAL (18, 2),  
6     PRIMARY KEY (Id)  
7 );
```

# LES CONTRAINTES EN SQL

## Contrainte PRIMARY KEY (1/3)

- ▶ Une clé primaire est un champ dans une table qui identifie de manière unique chaque ligne/enregistrement dans une table de base de données. Les clés primaires doivent contenir des valeurs uniques. Une colonne de clé primaire ne peut pas avoir de valeur **NULL**.
- ▶ Une table ne peut avoir qu'une seule clé primaire, qui peut consister en un ou plusieurs champs. Lorsque plusieurs champs sont utilisés comme clé primaire, ils sont appelés clé composite.
- ▶ **Exemple 1:**

Voici la syntaxe pour définir l'attribut **Id** en tant que clé primaire dans une table **Employés**

```
1 CREATE TABLE Employes(  
2     Id NUMBER          NOT NULL PRIMARY KEY,  
3     Nom VARCHAR2 (20)  NOT NULL,  
4     Age  NUMBER        NOT NULL,  
5     Salaire  NUMBER(18, 2),  
6     );  
7
```

# LES CONTRAINTES EN SQL

## Contrainte PRIMARY KEY (2/3)

### ► Exemple 2 :

Pour créer une contrainte **PRIMARY KEY** sur les colonnes 'Id' et 'Nom', utilisez la syntaxe SQL suivante :

```
1 CREATE TABLE Employes(  
2     Id NUMBER          NOT NULL,  
3     Nom VARCHAR2 (20)   NOT NULL,  
4     Age  NUMBER         NOT NULL,  
5     Salaire  NUMBER (18, 2),  
6     CONSTRAINT pk_Employees PRIMARY KEY (Id, Nom)  
7 );
```

### ► Exemple 3:

Pour définir une contrainte **PRIMARY KEY** sur la colonne 'Id' alors que la table Employés existe déjà, utilisez la syntaxe donnée ci-dessous:

```
1 ALTER TABLE Employes  
2     ADD CONSTRAINT pk_Employees PRIMARY KEY (Id); 27
```

# LES CONTRAINTES EN SQL

## Contrainte PRIMARY KEY (3/3)

### ► Exemple 5:

Vous pouvez supprimer les contraintes de clé primaire de la table avec la syntaxe donnée ci-dessous.



```
1ALTER TABLE Employes DROP PRIMARY KEY ;
```

Ou



```
1ALTER TABLE Employes DROP CONSTRAINT pk_Employees;
```



# LES CONTRAINTES EN SQL

## Contrainte FOREIGN KEY (1/3)

- ▶ Une clé étrangère est une clé utilisée pour relier deux tables. Ceci est parfois appelé aussi clé de référencement.
- ▶ Une clé étrangère est une colonne ou une combinaison de colonnes dont les valeurs correspondent à une clé primaire dans une autre table.
- ▶ La relation entre 2 tables correspond à la clé primaire dans l'une des tables avec une clé étrangère dans la seconde table.

# LES CONTRAINTES EN SQL

## Contrainte FOREIGN KEY (2/3)

- Considérez la structure des deux tables suivantes.

```
1 CREATE TABLE Employes(  
2     Id NUMBER          NOT NULL,  
3     Nom VARCHAR2 (20)  NOT NULL,  
4     Age  NUMBER        NOT NULL,  
5     Salaire  NUMBER (18, 2),  
6     CONSTRAINT pk_Employees PRIMARY KEY (Id)  
7 );
```

Ou

```
1 CREATE TABLE Conges(  
2     Id NUMBER          NOT NULL PRIMARY KEY,  
3     Date_debut  DATE,  
4     Date_fin   DATE,  
5     Id_Emp  NUMBER,  
6     FOREIGN KEY (Id_Emp) REFERENCES  
7     Employes(Id)  
8 );
```



```
1 CREATE TABLE Conges(  
2     Id NUMBER          NOT NULL,  
3     Date_debut  DATE  NOT NULL,  
4     Date_fin   DATE  NOT NULL,  
5     ID_Emp  NUMBER,  
6     PRIMARY KEY (Id),  
7     FOREIGN KEY (ID_Emp) REFERENCES Employes(Id)  
8 );
```

# LES CONTRAINTES EN SQL

## Contrainte FOREIGN KEY (3/3)

- Si la table **Conges** a déjà été créée et que la clé étrangère n'a pas encore été définie, utilisez la syntaxe suivante pour spécifier une clé étrangère en modifiant une table.



```
1 ALTER TABLE Conges
2   ADD CONSTRAINT fk_Conges
   FOREIGN KEY (ID_EMP) REFERENCES Employes(Id);
```

Pour supprimer une contrainte **FOREIGN KEY**, utilisez la syntaxe suivante :

```
1 ALTER TABLE Conges
2   DROP CONSTRAINT fk_Conges;
```