

LANGUAGE SQL

LDD

BTS DAKHLA

DAHAR RACHID

Nous avons installé la dernière séance **deux logiciels**:

- **SQL Server** qui représente le serveur de gestion de bases de données de Microsoft. C'est un serveur (comme Access , Oracle)
- Pour exploiter un serveur de bases de données, nous avons besoin d'une interface conviviale ce qu'offre le **SQL Server Management Studio**. C'est le client (Comme SQL Développer).

SQL : INTRODUCTION

Le langage SQL (Structured Query Language) est un langage informatique utilisé pour exploiter des bases de données. Il permet de façon générale la définition **LDD**, la manipulation **LMD** et le contrôle de sécurité de données **LCD**.

Dans la pratique, le langage SQL est utilisé pour créer des tables (LDD), ajouter des enregistrements sous forme de lignes, interroger une base de données, la mettre à jour (LMD) ou encore gérer les droits d'utilisateurs de cette base de données (LCD). Il est bien supporté par la très grande majorité des systèmes de gestion de base de données (SGBD). Crée au début des années 1970 par Donald D. Chamberlin et Raymond F. Boyce, tous deux chez IBM, le langage SQL est aujourd'hui reconnu comme une norme internationale

LE LANGAGE SQL-LDD : CRÉATION DE BASE DE DONNÉES

Commençons par l'instruction CREATE DATABASE la plus simple possible. Il s'agit simplement de [CREATE DATABASE nom_base](#).

L'instruction suivante créera une base de données appelée MyDatabase avec les fichiers physiques dans l'emplacement de fichier par défaut, le login que vous utilisez sera le propriétaire et avec toutes les valeurs par défaut qui sont configurées dans la base de données modèle.

CREATE DATABASE MyDatabase

C'était très facile, mais nous allons probablement vouloir un peu plus de contrôle sur notre configuration.

OÙ SONT STOCKÉS LES FICHIERS DE LA BASES DE DONNÉES ?

A ne pas confondre: Fichiers de la base de données et fichiers SQL.

- Vos fichiers SQL, vous pouvez les stocker où vous voulez: sur votre disque, clé USB, etc..
- La commande **CREATE DATABASE nomBD** implique la création de deux fichiers sur votre disque dur
 - Un fichier nomBD.mdf
 - Un fichier nomBD.ldf
- Les données sont stockées dans un fichier MDF, toutes les transactions sont stockées dans un fichier LDF. (journal des logs nécessaires pour récupérer la base de données à partir d'un certain point).

LE LANGAGE SQL-LDD : CRÉATION DE BASE DE DONNÉES

```
CREATE DATABASE Sales
ON
( NAME = Sales_dat,
  FILENAME = 'D:\Databases\saledat.mdf',
  SIZE = 10,
  MAXSIZE = 50,
  FILEGROWTH = 5%)
LOG ON
( NAME = Sales_log,
  FILENAME = 'D:\Databases\salelog.ldf',
  SIZE = 5MB,
  MAXSIZE = 25MB,
  FILEGROWTH = 5MB );
```

Cet exemple crée la base de données Sales. Comme le mot clé PRIMARY n'est pas utilisé, le premier fichier (Sales_dat) devient le fichier primaire. Comme ni MB ni KB ne sont spécifiés dans le paramètre SIZE du fichier Sales_dat, celui-ci utilise MB mégaoctets. Le fichier Sales_log est alloué en mégaoctets car le suffixe MB est explicitement indiqué dans le paramètre SIZE.

- La taille par défaut de la base de données modèle est de 8 Mo (à partir de SQL Server 2016 (13.x)) ou de 1 Mo (pour les versions antérieures).
- MAXSIZE : Ne pas inclure de décimale. Si maxsize n'est pas spécifié, le fichier grandit jusqu'à ce que le disque soit plein. MAXSIZE est une valeur entière. Pour les valeurs supérieures à 2147483647, utilisez des unités plus grandes.

UNLIMITED Spécifie que le fichier se développe jusqu'à ce que le disque soit plein. Dans SQL Server, un fichier journal spécifié avec une croissance illimitée a une taille maximale de 2 To, et un fichier de données a une taille maximale de 16 To.

Si **FILEGROWTH** n'est pas spécifié, les valeurs par défaut sont : 64 Données 64 Mo. Fichiers journaux 64 Mo.

LE LANGAGE SQL-LDD : CRÉATION D'UNE TABLE

- La commande CREATE TABLE sert à spécifier une nouvelle relation en lui attribuant un **nom**, des **attributs** et des **contraintes**
- Les attributs doivent être en premier. Chaque attribut doit avoir un **nom** , un **type de données** (pour indiquer le domaine auquel appartiennent ses valeurs) et des **contraintes** éventuelles (NOT NULL par exemple)
- Les **contraintes de clé , d'intégrité de l'entité** et **d'intégrité référentielle** peuvent être spécifiées dans l'instruction **CREATE TABLE** après la déclaration des attributs ou être ajoutées plus tard à l'aide de la commande **ALTER TABLE**

LE LANGAGE SQL-LDD : CRÉATION D'UNE TABLE

```
CREATE TABLE nom_table(  
    nom_colonne      type_donnee_colonne [definition_contrainte_colonne],  
    nom_colonne      type_donnee_colonne [definition_contrainte_colonne],  
    ....  
    [definition_contrainte_table],  
    ....  
    [definition_contrainte_table]  
);
```

TYPE DE DONNÉES ET DOMAINES DES ATTRIBUTS

□ SQL offre divers **types de données** de base , dans la déclaration d'une colonne d'une table . On citera les principaux existant en **SQL Server**:

- **SMALLINT** : entier signés courts (ex 16 bits)
- **INTEGER (ou INT)** : entier signés longs (ex 32 bits)
- **NUMERIC (p,q)** : nombre de décimaux de p chiffres dont q après la point décimal, s'elle n'est pas mentionnée la valeur de q est 0
- **DECIMAL (p,q)** : nombre de décimaux d'au moins p chiffres dont q après la point décimal, s'elle n'est pas mentionnée la valeur de q est 0
- **FLOAT (p) ou FLOAT** : nombre en virgules flottante d'au moins p bit significatifs

TYPE DE DONNÉES ET DOMAINES DES ATTRIBUTS

- **CHAR (n)** : chaîne de longueur fixe de n caractère , La chaîne est complétée par des espaces si elle est plus petite que la taille déclarée
- **VARCHAR (n)** : chaîne de longueur variables d'au plus n caractères
- **DATE** : dates (année , mois et jours)
- **TIME** : instants (heure , minute , seconde)
- **TIMESTAMP** : date + temps exemple: ‘2022-09-25 19:15:45’

En plus de ces types de base , il est possible de définir des types :

CREATE TYPE MONTANT FROM DECIMAL(10,2);

CONTRAINTES D'INTÉGRITÉ

- Les contraintes de base (contraintes de colonnes) permettant de spécifier différentes contraintes portant sur un seul attribut, lors de ma création d'une table . Parmi celle-ci nous avons :
 - **Contraintes sur les attributs et de leurs valeurs par défaut** (NOT NULL , DEFAULT , CHECK)
 - **Contrainte de clé (Unicité de l'attribut)** : UNIQUE ou PRIMARY KEY
 - **Contrainte référentielle** : FOREIGN KEY

RÈGLE RELATIVES AUX CONTRAINTES

- Vous pouvez créer une contrainte à différents instants:
 - Lors de la création de la table
 - Après la création de la table
- Définissez une contrainte au **niveau colonne** ou **au niveau table**.

Niveau Colonne :

```
CREATE TABLE Products (
```

```
    ProductID INT CONSTRAINT pk_products_pid PRIMARY KEY,  
    ProductName VARCHAR(25)
```

```
);
```

Niveau Table :

```
CREATE TABLE Products (
```

```
    ProductID INT,
```

```
    ProductName VARCHAR(25),
```

```
    CONSTRAINT pk_products_pid PRIMARY KEY(ProductID)
```

```
);
```

CONTRAINTE SUR LES ATTRIBUTS ET DE LEURS VALEURS PAR DÉFAUT

Valeur nulle impossible :

- SQL permet des valeurs NULL pour les attributs , et il est possible de spécifier une contrainte NOT NULL pour interdire cette valeur.
- Cette contrainte est spécifier implicitement pour les attributs qui font partie de la **clé primaire**, mais peut être spécifiée pour n’importe quel autre attribut dont il est obligatoire que les valeur ne soient pas NULL

EXEMPLE

```
CREATE TYPE montant FROM decimal(6,2)

CREATE TABLE Employe (
    Matricule INT NOT NULL,
    Nom VARCHAR(20) NOT NULL,
    Prenom VARCHAR(20),
    Sexe CHAR(1),
    Ville VARCHAR(20),
    Email VARCHAR(25) ,
    Salaire montant -- il faut créer le type montant
);
```

CONTRAINTE SUR LES ATTRIBUTS ET DE LEURS VALEURS PAR DÉFAUT

Valeur par Défaut :

- Il est possible de définir la valeur par défaut d'un attribut en ajoutant la clause **DEFAULT <valeur>** à sa définition.
- La valeur par défaut est alors incluse dans tout nouveau tuple pour lequel il n'a pas été explicitement fourni de valeur.
- En absence de clause défaut, la valeur par défaut est NULL pour les attribut sur lesquels ne porte pas de contrainte NOT NULL.

EXEMPLE

```
CREATE TABLE Employe (
    Matricule INT NOT NULL,
    Nom VARCHAR(20) NOT NULL,
    Prenom VARCHAR(20),
    Sexe CHAR(1) DEFAULT 'M',
    Ville VARCHAR(20) DEFAULT 'DAKHLA',
    Email VARCHAR(25),
    Salaire montant -- il faut créer le type montant
);
```

```
CREATE TABLE Employe (
    Matricule INT NOT NULL ,
    Nom VARCHAR(20) NOT NULL,
    Prenom VARCHAR(20),
    Sexe CHAR(1) Constraint DF_Sexe DEFAULT 'M',
    Ville VARCHAR(20) Constraint DF_Ville DEFAULT 'DAKHLA',
    Email VARCHAR(25),
    Salaire montant -- il faut créer le type montant
);
```

CONTRAINTE SUR LES ATTRIBUTS ET DE LEURS VALEURS PAR DÉFAUT

La contrainte CHECK impose un domaine de valeurs ou une condition simple ou complexe entre colonnes.

CHECK (note BETWEEN 0 AND 20),

CHECK (grade='Copilote' OR grade='Commandant')

EXEMPLE

```
CREATE TABLE Employe (
    Matricule INT NOT NULL,
    Nom VARCHAR(20) NOT NULL,
    Prenom VARCHAR(20) ,
    Sexe CHAR(1) DEFAULT 'M' CHECK (Sexe IN ('M','F')),
    Ville VARCHAR(20) DEFAULT 'DAKHLA',
    Email VARCHAR(25) ,
    Salaire montant CHECK (Salaire BETWEEN 0 AND 10000)
);
```

```
CREATE TABLE Employe (
    Matricule INT NOT NULL,
    Nom VARCHAR(20) NOT NULL,
    Prenom VARCHAR(20) ,
    Sexe CHAR(1) Constraint DF_Sexe DEFAULT 'M' Constraint CK_Sexe CHECK (Sexe IN
        ('M','F')),
    Ville VARCHAR(20) Constraint DF_Ville DEFAULT 'DAKHLA',
    Email VARCHAR(25),
    Salaire montant Constraint CK_Salaire CHECK (Salaire BETWEEN 0 AND 10000)
);
```

CONSTRAINTS DE CLÉ

La Clause **PRIMARY KEY** désigne le ou les attributs qui forment la clé primaire d'une relation lors de la création d'une table

```
CREATE TABLE Employe (
    Matricule INT PRIMARY KEY,
    Nom VARCHAR(20) NOT NULL,
    Prenom VARCHAR(20) ,
    Sexe CHAR(1) DEFAULT 'M' CHECK (Sexe IN ('M','F')),
    Ville VARCHAR(20) DEFAULT 'DAKHLA',
    Email VARCHAR(25),
    Salaire montant CHECK (Salaire BETWEEN 0 AND 10000)
    -- il faut créer le type montant
);
```

L'OPTION IDENTITY

l'option IDENTITY pour incrémenter automatiquement la clé primaire. L'avantage d'avoir une telle option est que la clé primaire ne sera jamais dupliquée. On diminue les erreurs . La clé est auto-générée
L'option IDENTITY convient surtout pour les tables ayant comme clé primaire un numéro séquentiel .

```
CREATE TABLE Carburant (
    IdType INT PRIMARY KEY IDENTITY(1,1),
    TypeCarb VARCHAR(20) NOT NULL,
);
```

CONSTRAINTES DE CLÉ

La clause **UNIQUE** Indique que les valeurs saisies pour les colonnes (attributs) doivent être uniques, ce qui veut dire pas de doublons.

Exemple: Le numéro d'assurance sociale, le numéro d'un permis de conduire, un courriel.

```
CREATE TABLE Employe (
    Matricule INT PRIMARY KEY,
    Nom VARCHAR(20) NOT NULL,
    Prenom VARCHAR(20) ,
    Sexe CHAR(1) DEFAULT 'M' CHECK (Sexe IN ('M','F')),
    Ville VARCHAR(20) DEFAULT 'DAKHLA',
    Email VARCHAR(25) UNIQUE,
    Salaire montant CHECK (Salaire BETWEEN 0 AND 10000)
    -- il faut créer le type montant
);
```

CONTRAINTE D'INTÉGRITÉ RÉFÉRENTIELLE

- On parle d'intégrité référentielle lorsque les valeurs d'une ou plusieurs colonnes d'une table sont déterminées ou font référence à des valeurs d'une colonne d'une autre table.
- Dans notre exemple ci-après, les valeurs de la colonne codeequipe de la table Joueurs font référence aux valeurs de la colonne codeequipe de la table Equipes
- La table Joueurs ne contient aucun codeequipe qui n'est pas dans la table Equipes.

Table joueurs

NUMJUEUR	NOM	PRENOM	CODEEQUIPE
1	PRICE	CAREY	MTL
2	MARKOV	ANDRÉ	MTL
3	SUBBAN	KARL	MTL
4	PATIORETTY	MAX	MTL
10	HAMOND	ANDREW	OTT
6	STONE	MARC	OTT
9	TURIS	KYLE	OTT
7	GALLAGHER	BRANDON	MTL
8	TANGUAY	ALEX	AVL
11	THOMAS	BIL	AVL

Table Equipes

CODEEQUIPE	NOMEQUIPE	VILLE	NBCOUPE
1 MTL	LES CANADIENS DE MONTRÉAL	MONTRÉAL	24
2 TOR	LES MAPLE LEAFS	TORONTO	22
3 OTT	LES SÉNATEURS	OTTAWA	4
4 AVL	LES AVALANCHES	COLORADO	2
5 VAN	LES CANUKS	VANCOUVER	1
6 BRU	LES BRUNS DE BOSTON	BOSTON	13

référence

La table EQUIPES doit être créée en premier

```
CREATE table EQUIPES(
    codeequipe CHAR(3) CONSTRAINT pk_equipe PRIMARY KEY,
    nomequipe VARCHAR(50) NOT NULL,
    Ville VARCHAR(40),
    nbcouipes NUMERIC(2,0) CONSTRAINT ck_nbcoupe CHECK (nbcouipes > =0)
);
```

On crée la table JOUEURS après

```
CREATE TABLE Joueurs(
    numjoueur NUMERIC(3,0) CONSTRAINT pk_joueurs PRIMARY KEY,
    nom VARCHAR(30) NOT NULL,
    prenom VARCHAR(30),
    codeequipe CHAR(3),
    CONSTRAINT fk_Joueurs_equipes FOREIGN KEY (codeequipe) REFERENCES
        equipes(codeequipe)
);
```

- Les attributs codeequipe des deux tables sont de même type et de même longueur: CHAR(3). C'est une obligation.
- Le mot réservé **REFERENCES** indique la colonne (attribut) référencée de la table de la clé primaire. C'est pourquoi la table de la clé primaire doit être créée en premier. On ne peut pas référencer quelque chose qui n'existe pas.

CONTRAINTE FOREIGN KEY : MOTS CLÉS : ON DELETE

- **ON DELETE CASCADE** : supprime les lignes dépendantes dans la table enfant lorsqu'une ligne de la table parent est supprimée.
- **ON DELETE SET NULL**: convertit les valeurs des clés étrangères dépendantes en valeurs NULL. Cette contrainte ne peut être exécutée que si toutes les colonnes de clé étrangère de la table cible acceptent des valeurs NULL.
- **SET DEFAULT** place la valeur par défaut (qui suit ce paramètre) dans la ligne de la table étrangère en cas d'effacement d'une valeur correspondant à la clé

On crée la table JOUEURS après

```
CREATE TABLE Joueurs(
    numjoueur NUMERIC(3,0) CONSTRAINT pk_joueurs PRIMARY KEY,
    nom VARCHAR(30) NOT NULL,
    prenom VARCHAR(30),
    code CHAR(3),
    CONSTRAINT fk_Joueurs_equipes FOREIGN KEY (code) REFERENCES equipes(codeequipe) ON
    DELETE CASCADE
);
```

CONTRAINTE FOREIGN KEY : MOTS CLÉS : ON UPDATE

- **ON UPDATE CASCADE** : Les valeurs associées dans la table enfant seront également mises à jour.
- **ON UPDATE SET NULL**: convertit les valeurs des clés étrangères dépendantes en valeurs NULL. Cette contrainte ne peut être exécutée que si toutes les colonnes de clé étrangère de la table cible acceptent des valeurs NULL.
- **ON UPDATE SET DEFAULT** Les valeurs associées dans la table enfant seront définies par la valeur par défaut spécifiée dans la définition de la colonne.

On crée la table JOUEURS après

```
CREATE TABLE Joueurs(
    numjoueur NUMERIC(3,0) CONSTRAINT pk_joueurs PRIMARY KEY,
    nom VARCHAR(30) NOT NULL,
    prenom VARCHAR(30),
    code CHAR(3),
    CONSTRAINT fk_Joueurs_equipes FOREIGN KEY (code) REFERENCES equipes(codeequipe) ON
    DELETE CASCADE ON UPDATE CASCADE
);
```

MODIFICATIONS STRUCTURELLES (ALTER TABLE)

➤ Ajout de colonnes

la directive **ADD** de l'instruction ALTER TABLE permet d'ajouter

```
ALTER TABLE Joueurs ADD NbButs int DEFAULT 0;
```

➤ Renommer des colonnes

```
EXEC sp_rename 'Joueurs.NbButs', 'Goals';
```

➤ Modifier le type des colonnes

```
ALTER TABLE Joueurs ALTER COLUMN Goals decimal(3,0);
```

➤ Modifier le type des colonnes

L'option **DROP** de l'instruction ALTER TABLE permet de supprimer une colonne.

```
ALTER TABLE Joueurs DROP COLUMN Goals
```

ALTER TABLE : SUPPRESSION/AJOUT CONTRAINTE

- suppression de contrainte

```
ALTER TABLE Joueurs DROP CONSTRAINT fk_Joueurs_equipes;
```

- Ajout de contrainte

- ```
ALTER TABLE Joueurs ADD CONSTRAINT fk_Joueurs_equipes FOREIGN KEY (code) REFERENCES equipes(codeequipe) ON DELETE CASCADE ON UPDATE CASCADE;
```
- ```
ALTER TABLE Joueurs ADD CONSTRAINT CK_Goals CHECK (Goals >= 0)
```

- Supprimer une table

```
DROP TABLE Joueurs
```