Chapitre 6:
Le langage SQL
Structured Query Language

Introduction

SQL vs Algèbre Relationnelle

Historique

Langage de Définition de Données LDD

Langage de Manipulation de Données LMD

Langage d'Interrogation des Données LID

Langage de Contrôle de Données LCD

INTRODUCTION

- SQL (**Structured Query Language**) est un **langage informatique normalisé** qui permet d'interagir avec des bases de données relationnelles et de les exploitées à travers des requêtes;
- Quand on souhaite interroger une base de données, on utilise des commandes sous forme de requêtes (récupération, modification, ajout, suppression ...).

INTRODUCTION

- Les requêtes SQL sont classifiées selon les langages (sous-langage SQL) suivants :
  - ✓ **Langage de Définition de Données LDD** (en anglais Data Definition Language DDL) permettant de définir les schémas des BD;
  - ✓ **Langage de Manipulation de Données LMD** (en anglais Data Manipulation Language DML) permettant de mettre à jour les données d'une BD (Ajout, Modification, Suppression);
  - ✓ **Langage d'Interrogation des Données LID** (en anglais Data Query Language DQL) permettant d'exprimer toutes les questions que l'on peut poser à une base de données;
  - ✓ **Langage de Contrôle de Données LCD** (en anglais Data Control Language DCL) permettant de gérer les utilisateurs (création des utilisateurs et affectation de leurs droits) d'une BD et de contrôler l'exécution des transactions.

SQL VS ALGÈBRE RELATIONNELLE

- L'algèbre relationnelle est un **langage formelle (théorique)** de manipulation des données qui exprime une succession d'opérations effectuées sur les relations ;
- L'algèbre relationnelle ne peut pas être **compris directement par les SGBDR** ;
- SQL est un **langage pratique (compris et exécutée directement par les SGBDR) fondé sur l'algèbre relationnelle**.

HISTORIQUE

- **Apparition du langage SQL:**
  - ✓ **Années 70** : la société IBM avec le SGBD System-R a implanté le modèle relationnel au travers du langage SEQUEL (Structured English as QUEry Language) ;
  - ✓ **Début des années 80** : ORACLE propose la première version du langage SQL basé sur le langage SEQUEL;
- **Normalisation du langage SQL :**
  - ✓ **1987**: Apparition de la première norme appelé **SQL1** par ISO (**I**nternational **S**tandard **O**rganisation) et ANSI (**A**merican **N**ational **S**tandard **I**nstitute) :
    - ➢ Types de données simples (entiers, réels, chaînes de caractères de taille fixe).

HISTORIQUE

- **Normalisation du langage SQL: (Suite)**
  - ✓ **1992**: La norme SQL1 a été révisé par ISO et ANSI ce qui a donné comme résultat la deuxième norme appelé SQL-92 ou SQL2:
    - ➢ Types de données plus riches: dates, chaînes de caractères de taille variable…;
    - ➢ Différents types de jointures: jointure naturelle, jointure externe;
    - ➢ Possibilité de renommage des attributs;
    - ➢ Etc.

HISTORIQUE

- **Normalisation du langage SQL: (Suite)**
  - ✓ **Depuis 1999** : introduction de la norme SQL3 par ISO et ANSI qui comporte de nombreuses fonctionnalités:
    - ➢ Concepts le l'orienté objets;
    - ➢ Entrepôts de données;
    - ➢ Accès à des sources non SQL;
    - ➢ Réplication des données;
    - ➢ etc.

Langage de Définition de Données LDD

Introduction
Création d'une base de données
Suppression d'une BD et commentaires
Création d'une table
Principaux types de base
- ✓ Alphanumériques
- ✓ Numériques: les entiers
- ✓ Numériques: les décimaux
- ✓ Dates et heures
- ✓ Données binaires
- ✓ Énumérations

Contraintes d'intégrités
- ✓ La contrainte NOT NULL
- ✓ La contrainte DEFAULT
- ✓ La contrainte UNIQUE
- ✓ La contrainte CHECK
- ✓ La contrainte PRIMARY KEY
- ✓ La contrainte FOREIGN KEY

INTRODUCTION

- Le **L**angage de **D**éfinition de **D**onnées **LDD** permet de définir le schéma d'une base de données c'est-à-dire les schémas des tables appartenant à la base de données (DataBase);
- La définition du schéma d'une base de données consiste à créer la base de données et à définir les schémas de toutes les tables (relations) appartenant à la base de données ainsi que les contraintes d'intégrités structurelles;
- Concepts de base du langage SQL :
  - ✓ **Relation** → **Table**
  - ✓ **Attribut** → **Colonne**
  - ✓ **Tuple** → **Ligne**
  - ✓ **Schéma** → **Structure**

CRÉATION D'UNE BASE DE DONNÉES

- Dans le cadre de ce cours nous utiliserons le SGBDR MySQL qui est basé sur l'architecture client-serveur;
- Au sein d'un serveur on peut créer autant de BD que l'on veut, chaque BD doit avoir son propre nom;
- Dans une BD on peut trouvés plusieurs tables, une table appartienne à une seule BD, pour créer une BD exécuter une requête contenant la commande CREATE DATABASE; Syntaxe: **CREATE DATABASE** nom_base**;**

Exemple: **CREATE DATABASE** Gestion_notes**;**

Remarque: Chaque requête SQL doit se terminer par un point virgule « ; ».

SUPPRESSION D'UNE BD ET COMMENTAIRES

- Pour supprimer une BD exécuter une requête contenant la commande DROP DATABASE; Syntaxe: **DROP DATABASE** nom_base**;**

Exemple: **DROP DATABASE** Gestion_notes; #Suppression de la BD Gestion_notes

- Les symboles dièse « # » et « -- » permettent de faire un commentaire jusqu'à la fin de la ligne;
- Pour avoir un commentaire sur plusieurs lignes on utilise « /* mon commentaire */ » ;
- Un commentaire sur plusieurs lignes a l'avantage de pouvoir indiquer le début et la fin du commentaire, ce qui rend possible de l'utiliser au corps d'une requête SQL sans problème.

CRÉATION D'UNE TABLE

- Pour créer une table exécuter une requête contenant la commande CREATE TABLE;

Syntaxe:

```
CREATE TABLE nom_table
        (
          nom_col_1 type_col_1,
                ... ,
        nom_col_n type_col_n
        );
```

Exemple:

```
CREATE TABLE Matiere
        (
        CodeMat Integer ,
        Intitule VarChar(50)
        );
```

PRINCIPAUX TYPES DE BASE

- Pour décrire les colonnes d'une table, MySQL fournit les types prédéfinis suivants:
  - ✓ **Alphanumériques:** CHAR, VARCHAR, TINYTEXT, TEXT, MEDIUMTEXT, LONGTEXT;
  - ✓ **Numériques:** TINYINT, SMALLINT, MEDIUMINT, INT, INTEGER, BIGINT, FLOAT, DOUBLE, REAL, DECIMAL, NUMERIC;
  - ✓ **Date/Heure:** DATE, DATETIME, TIME, YEAR, TIMESTAMP;
  - ✓ **Données binaires:** BLOB, TINYBLOB, MEDIUMBLOB, LONGBLOB;
  - ✓ **Énumérations:** ENUM, SET.

ALPHANUMÉRIQUES

Type	Description	Longueur Maximale
CHAR(n)	Chaîne fixe de n caractères (n octets)	n <=2^8–1 (255 octets)
VARCHAR(n)	Chaîne variable de n caractères (n octets)	n <= 2^{16}–1 (65 535 octets)
TINYTEXT(n)	Chaîne fixe de n caractères (n octets)	n <=2^8–1 (255 octets)
TEXT(n)	Chaîne fixe de n caractères (n octets)	n <= 2^{16}–1 (65 535 octets)
MEDIUMTEXT(n)	Chaîne fixe de n caractères (n octets)	n <= 2^{24}–1 (16 Mo)
LONGTEXT(n)	Chaîne fixe de n caractères (n octets)	n <= 2^{32} –1 (4,29 Go)

- Pour les chaînes fixe les valeurs sont stockées en ajoutant des blancs si sa taille est inférieure à **n**. Ces blancs ne seront pas considérés après extraction à partir de la table;
- Pour les chaînes variable les valeurs sont stockées sans l'ajout de blancs, on stocke la chaîne plus la longueur de la chaîne stockées.

NUMÉRIQUES: LES ENTIERS

Type	Valeurs minimales et maximales
INTEGER[(n)] [UNSIGNED] [ZEROFILL]	Entier (sur 4 octets): signé de -2^{31} à $2^{31}-1$, non signé de 0 à $2^{32}-1$.
TINYINT [(n)] [UNSIGNED] [ZEROFILL]	Entier (sur 1octet): signé de -2^7 à 2^7-1, non signé de 0 à 2^8-1.
SMALLINT[(n)] [UNSIGNED] [ZEROFILL]	Entier (sur 2 octets): signé de -2^{15} à $2^{15}-1$, non signé de 0 à $2^{16}-1$.
MEDIUMINT[(n)] [UNSIGNED] [ZEROFILL]	Entier (sur 3 octets): signé de -2^{23} à $2^{23}-1$, non signé de 0 à $2^{24}-1$.
BIGINT[(n)] [UNSIGNED] [ZEROFILL]	Entier (sur 8 octets): signé de -2^{63} à $2^{63}-1$, non signé de 0 à $2^{64}-1$.

- La directive UNSIGNED permet de considérer seulement des valeurs positives;
- La directive ZEROFILL complète par des zéros à gauche une valeur (par exemple : soit un INTEGER(3) contenant valeur 4, si ZEROFILL est appliqué, la valeur extraite sera «004»).

NUMÉRIQUES: LES DÉCIMAUX

Type	Valeurs minimales et maximales
FLOAT[(n[,p])] [UNSIGNED] [ZEROFILL]	Flottant en précision simple: signé de $-3.4\ 10^{+38}$ à $-1.1\ 10^{-38}$, 0 . non signé de $1.1\ 10^{-38}$ à $3.4\ 10^{+38}$.
DOUBLE[(n[,p])] [UNSIGNED][ZEROFILL]	Flottant en précision double: signé de $-1.7\ 10^{+308}$ à $-2.2\ 10^{-308}$, 0 . non signé de $2.2\ 10^{-308}$ à $1.7\ 10^{+308}$.
DECIMAL(n,p) [UNSIGNED] [ZEROFILL]	Décimal à virgule fixe (par défaut **n** vaut 10, **p** vaut 0). Stocké sous frome de chaîne: chaque chiffre, le signe «–», la virgule «,» occupe un caractère (octet), mais le signe «–» et la virgule «,» ne sont pas comptés dans **n**.

- Dans toute instruction SQL, écrivez la virgule avec un point (7/2 retourne 3.5).

DATES ET HEURES

Type	Description	Taille et format
DATE	Dates du 1er janvier de l'an 1000 au 31 décembre 9999	Sur 3 octets. L'affichage est au format 'YYYY-MM-DD'
DATETIME	Dates et heures (de 0 h de la première date à 23 h 59 minutes 59 secondes de la dernière date)	Sur 8 octets. L'affichage est au format 'YYYY-MM-DD HH:MM:SS'
YEAR[(2|4)]	Année de 1901 à 2155	Sur 1 octet. L'année est considérée sur 2 ou 4 positions (4 par défaut). Le format d'affichage est 'YYYY'.
TIME	Heures de -838 h 59 minutes 59 secondes à 838 h 59 minutes 59 secondes	L'heure au format 'HHH:MM:SS' sur 3 octets
TIMESTAMP	Instants du 1er Janvier 1970, 0 h 0 minute 0 seconde à 19 janvier 2038 à 3 h 14 min 7 s	Sur 4 octets (au format 'YYYY-MM-DD HH:MM:SS')

DONNÉES BINAIRES

- Les types BLOB (Binary Large OBject) permettent de stocker des données non structurées comme le multimédia (images, sons, vidéo, etc.). Ces types sont traités comme des flots d'octets sans jeu de caractère associé.

Type	Description	Taille
TINYBLOB(n)	Flot de n octets	n <= 2^8-1 (255 octets)
BLOB(n)	Flot de n octets	n <= 2^{16}-1 (65 535 octets)
MEDIUMBLOB(n)	Flot de n octets	n <= 2^{24}-1 (16 Mo)
LONGBLOB(n)	Flot de n octets	n <= 2^{32} -1 (4,29 Go)

ÉNUMÉRATIONS

Type	Nombre maximal de valeurs
ENUM('valeur1','valeur2',... 'valeur65535')	Liste de 65 535 valeurs au maximum
SET('valeur1','valeur2',... 'valeur64')	Liste de 64 valeurs au maximum

- **SET** et **ENUM** sont des types **propres à MySQL:**
  - ✓ **ENUM**: on ne peut choisir qu'une seule valeur parmi la liste;
  - ✓ **SET**: on peut choisir plusieurs valeurs parmi la liste.

CONTRAINTES D'INTÉGRITÉS

- Les Contraintes d'intégrités sont les règles que l'on peut demander au SGBD de garantir (vérification automatique par le SGBD);
- Les contraintes sur une colonne sont :
  - ✓ NOT NULL;
  - ✓ DEFAULT ;
  - ✓ UNIQUE ;
  - ✓ CHEK(Condition) ;
  - ✓ PRIMARY KEY ;
  - ✓ FOREIGN KEY.

LA CONTRAINTE NOT NULL

- La contrainte NOT NULL est une contrainte d'intégrité imposant qu'une colonne d'une table doit comporter une valeur non nulle (différente de nulle);
- Pour appliquer cette contrainte utiliser le mot clé NOT NULL pendant la définition des colonnes de la table;

Syntaxe:

```
nom_col  type_col  NOT NULL,
```


LA CONTRAINTE NOT NULL

Exemple:

```
CREATE TABLE Etudiant
        (
        NumInscrEt Integer NOT NULL,
        CinEt Char(7) NOT NULL,
        NomEt VarChar(50) NOT NULL,
        PrenomEt VarChar(50) NOT NULL,
        AdrEt VarChar(80),
        TeleEt Char(10)
        );
```


LA CONTRAINTE DEFAULT

- La contrainte DEFAULT est une contrainte d'intégrité qui s'applique lorsqu'une valeur d'une colonne n'est pas renseignée, il consiste à attribuer automatiquement une valeur à une colonne.
- Pour appliquer cette contrainte utiliser le mot clé DEFAULT suivi de la valeur par défaut;

Syntaxe:

```
nom_col  type_col  DEFAULT   'Valeur_par_defaut',
```


LA CONTRAINTE DEFAULT

Exemple:

```
CREATE TABLE Etudiant
        (
        NumInscrEt Integer NOT NULL,
        CinEt Char(7) NOT NULL,
        Nationalite VARCHAR (30) NOT NULL DEFAULT 'Marocaine' ,
        NomEt VarChar(50) NOT NULL,
        PrenomEt VarChar(50) NOT NULL,
        AdrEt VarChar(80),
        TeleEt Char(10)
        );
```

LA CONTRAINTE UNIQUE

- La contrainte UNIQUE est une contrainte d'intégrité imposant que toute les valeurs de la colonne soient distinctes (impossibles de trouvé deux lignes ou plus ayant les mêmes valeurs pour cette colonne);
- Pour appliquer cette contrainte utiliser le mot clé UNIQUE pendant la définition des colonnes de la table;

Syntaxe:

```
nom_col  type_col  UNIQUE,
```


LA CONTRAINTE UNIQUE

Exemple:

```
CREATE TABLE Etudiant
        (
        NumInscrEt Integer UNIQUE,
        CinEt Char(7) UNIQUE,
        Nationalite VARCHAR (30) NOT NULL DEFAULT 'Marocaine' ,
        NomEt VarChar(50) NOT NULL,
        PrenomEt VarChar(50) NOT NULL,
        AdrEt VarChar(80),
        TeleEt Char(10) UNIQUE
        );
```


LA CONTRAINTE CHECK

- La norme SQL2 comprend une contrainte d'intégrité CHECK (condition) imposant qu'une colonne d'une table doit comporter des valeurs vérifiant la condition;
- Pour appliquer cette contrainte utiliser le mot clé CHECK puis exprimer la condition entre parenthèse pendant la définition des colonnes de la table, ou créer une contrainte avec le mot clé CONSTRAINT et exprimer la condition avec CHECK;

Syntaxes:

```
CHECK (condition sur une colonne)
```

Création d'une contrainte:

```
CONSTRAINT nom_contrainte CHECK (condition sur une colonne)
```

- **Pour MySQL Cette contrainte est prise en charge au niveau de la déclaration mais n'est pas encore opérationnelle.**

LA CONTRAINTE CHECK

Exemple:

```
CREATE TABLE Filiere
        (
        CodeFil Integer UNIQUE NOT NULL,
        IntituleFil VarChar(50),
        CONSTRAINT ck_intfil CHECK (IntituleFil IN ('GI', 'TM', 'TIMQ', 'GIM'))
        );
```

LA CONTRAINTE PRIMARY KEY

- La contrainte PRIMARY KEY déclare la clé primaire de la table, les colonnes clés primaires ne peuvent être ni nulles ni identiques;
- Pour appliquer cette contrainte utiliser le mot clé PRIMARY KEY pendant la définition des colonnes de la table, ou créer une contrainte avec le mot clé CONSTRAINT puis exprimer la clé primaire avec PRIMARY KEY;

Syntaxes:

```
nom_col  type_col PRIMARY KEY,
```

Création d'une contrainte:

```
CONSTRAINT nom_contrainte PRIMARY KEY (liste colonnes)
```


LA CONTRAINTE PRIMARY KEY

Exemple:

```
CREATE TABLE Etudiant
        (
        NumInscrEt Integer PRIMARY KEY,
        CinEt Char(7) UNIQUE NOT NULL,
        Nationalite VARCHAR (30) NOT NULL DEFAULT 'Marocaine' ,
        NomEt VarChar(50) NOT NULL,
        PrenomEt VarChar(50) NOT NULL,
        AdrEt VarChar(80),
        TeleEt Char(10) UNIQUE
        );
```


LA CONTRAINTE PRIMARY KEY

Exemple:

```
CREATE TABLE Matiere
        (
        CodeMat Integer ,
        Intitule VarChar(50),
        PRIMARY KEY (CodeMat)
        );
```


LA CONTRAINTE PRIMARY KEY

Exemple:

```
CREATE TABLE Note
        (
        NumInscrEt Integer,
        CodeMat Integer,
        Date_obtention DATE,
        Note DECIMAL(4,2) UNSIGNED ZEROFILL,
        CONSTRAINT  PK_Note PRIMARY KEY(NumInscrEt,CodeMat)
        );
```

LA CONTRAINTE FOREIGN KEY

- La contrainte FOREIGN KEY déclare la clé étrangère qui permet de mettre en place une contrainte de référence entre une ou plusieurs colonnes d'une table et les colonnes composantes les clés primaires des autres tables;
- Pour appliquer cette contrainte utiliser les mots clés FOREIGN KEY et REFERENCES ou créer une contrainte avec le mot clé CONSTRAINT puis exprimer la clé étrangère avec FOREIGN KEY;

Syntaxes: **FOREIGN KEY** (nom_col) **REFERENCES** nom_table(nom_col),

Création d'une contrainte:

CONSTRAINT nom_contrainte **FOREIGN KEY** (nom_col) **REFERENCES** nom_table(nom_col)

Remarque: la table référencée(clé primaire) doit être créée avant la table qui la référence(clé étrangère)

LA CONTRAINTE FOREIGN KEY

Exemple:

```
CREATE TABLE Note (
        NumInscrEt Integer,
        CodeMat Integer,
        Date_obtention DATE,
        Note DECIMAL(4,2) UNSIGNED ZEROFILL,
        PRIMARY KEY(NumInscrEt,CodeMat),
        FOREIGN KEY (NumInscrEt)REFERENCES Etudiant(NumInscrEt),
        FOREIGN KEY (CodeMat) REFERENCES Matiere(CodeMat)
        );
```


LA CONTRAINTE FOREIGN KEY

Exemple:

```
CREATE TABLE Note
        (
        NumInscrEt Integer,
        CodeMat Integer,
        Date_obtention DATE,
        Note DECIMAL(4,2) UNSIGNED ZEROFILL,
        CONSTRAINT  PK_Note PRIMARY KEY(NumInscrEt,CodeMat),
        CONSTRAINT FK_Note_Etu FOREIGN KEY (NumInscrEt)REFERENCES
        Etudiant(NumInscrEt),
        CONSTRAINT FK_Note_Mat FOREIGN KEY (CodeMat) REFERENCES
        Matiere(CodeMat)
        );
```


LA CONTRAINTE FOREIGN KEY

- Que se passe-t-il quand la violation d'une contrainte d'intégrité référentielle est détectée par le SGBD? par exemple supprimer (ou modifier le NumInscEt) d'un étudiant qui a des notes (référencé comme clé étrangère);
- Par défaut, le SGBD empêche l'exécution de cette requête en affichant un message d'erreur;
- Il est possible de demander la répercussion de cette mise à jour de manière à ce que la contrainte soit respectée;
- Les événements que l'on peut répercuter sont la modification ou la suppression de la ligne référencée, On les désignes par **ON UPDATE** et **ON DELETE** respectivement.

LA CONTRAINTE FOREIGN KEY

- Les répercussions que l'on peut demandées à MySQL sont:
  - ✓ **RESTRICT**: opération par défaut, MySQL empêche l'exécution de la requête ;
  - ✓ **NO ACTION:** même opération que **RESTRICT**;
  - ✓ **SET NULL:** met NULL les valeurs des lignes de table qui référence (met NULL les clés étrangères qui référence la valeur concernée);
  - ✓ **CASCADE:** appliquer la même opération aux valeurs des lignes de table qui référence (modification ou suppression des clés étrangères qui référence la valeur concernée).

LA CONTRAINTE FOREIGN KEY

Exemple:

```
CREATE TABLE Note
        (
        NumInscrEt Integer,
        CodeMat Integer,
        Date_obtention DATE,
        Note DECIMAL(4,2) UNSIGNED ZEROFILL,
        CONSTRAINT FK_Note_Etu FOREIGN KEY (NumInscrEt)REFERENCES
        Etudiant(NumInscrEt) ON DELETE SET NULL,
        CONSTRAINT FK_Note_Mat FOREIGN KEY (CodeMat) REFERENCES
        Matiere(CodeMat) ON DELETE SET NULL
        );
```


LA CONTRAINTE FOREIGN KEY

Exemple:

```
CREATE TABLE Note
        (
        NumInscrEt Integer,
        CodeMat Integer,
        Date_obtention DATE,
        Note DECIMAL(4,2) UNSIGNED ZEROFILL,
        CONSTRAINT FK_Note_Etu FOREIGN KEY (NumInscrEt)REFERENCES
        Etudiant(NumInscrEt) ON UPDATE CASCADE,
        CONSTRAINT FK_Note_Mat FOREIGN KEY (CodeMat) REFERENCES
        Matiere(CodeMat) ON UPDATE CASCADE
        );
```


LA CONTRAINTE FOREIGN KEY

Exemple:

```
CREATE TABLE Note
        (
        NumInscrEt Integer,
        CodeMat Integer,
        Date_obtention DATE,
        Note DECIMAL(4,2) UNSIGNED ZEROFILL,
        CONSTRAINT FK_Note_Etu FOREIGN KEY (NumInscrEt)REFERENCES
        Etudiant(NumInscrEt) ON UPDATE CASCADE ON DELETE RESTRICT,
        CONSTRAINT FK_Note_Mat FOREIGN KEY (CodeMat) REFERENCES
        Matiere(CodeMat) ON UPDATE CASCADE ON DELETE RESTRICT
        );
```